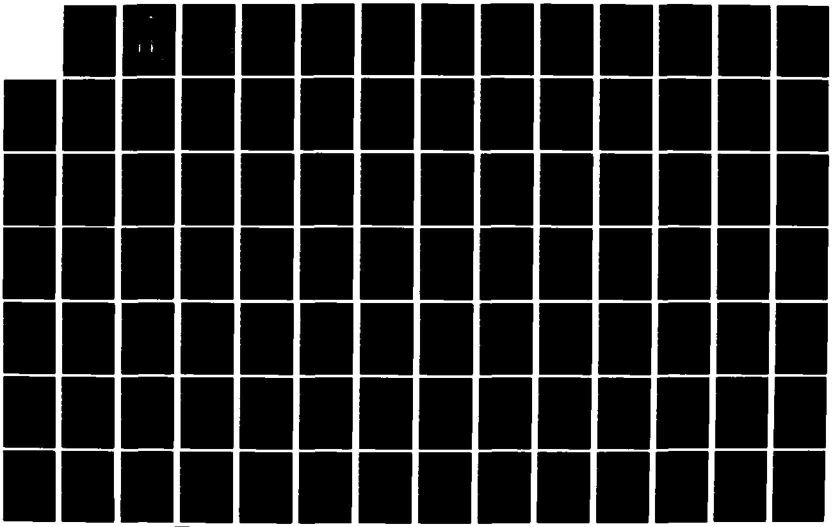
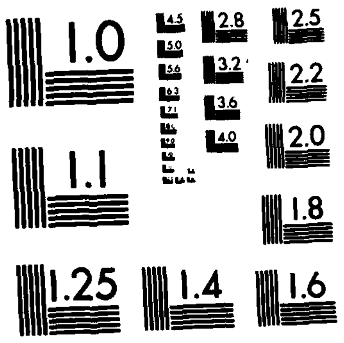
AD-A124 873 NUMERICAL COMPUTATION OF THE MATRIX RICCATI EQUATION 1/2
FOR HEAT PROPAGATION..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI..
UNCLASSIFIED P W SAGSTETTER DEC 82 AFIT/GCS/MA/82D-7 F/G 22/3 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/MA/82D-7

NUMERICAL COMPUTATION OF THE MATRIX RICCATI EQUATION FOR HEAT PROPAGATION DURING SPACE SHUTTLE REENTRY

THESIS

AFIT/GCS/MA/82D-7 Philip W. Sagstetter
Capt USAF

AFIT/GCS/MA/82D-7

NUMERICAL COMPUTATION OF THE MATRIX RICCATI EQUATION FOR HEAT PROPAGATION DURING SPACE SHUTTLE REENTRY

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

By

Philip W. Sagstetter, B.A.
Capt USAF
Graduate Computer Systems

December 1982

PREFACE

As an undergraduate of Physics, I often wondered how the professional engineers were actually using computers to solve all of those intimidating equations. Therefore, it was very satisfying for me to come to AFIT and to have the opportunity to study numerical analysis and computer systems. This thesis gave me a chance to apply that knowledge to an important problem and to make a contribution that might improve the capabilities of the Space Shuttle. This project also taught me some new things about matrix theory, and it was my first chance to work extensively with a main-frame scientific computer. It wasn't easy, but I am grateful for the experience.

Thanks are due to Captain James Hodge for his frequent assistance concerning the use of the HEATEST Program in particular and the CYBER computer in general. Captain David Audley was most helpful concerning the mathematical operations of the HEATEST Program. Of course, the overall guidance and patience of Dr. Dennis Quinn has been invaluable to the successful completion of this thesis.

Finally, I would like to apologize to my wife, Inok, for my frequent absence during her first 18 months of residence in the United States.

CONTENTS

List of Figures

List of Tables

NOMENCLATURE

A,B,C,D,T,U	Coefficient Matrices (L x L)
$C_A, C_B \ldots$	Material Specific Heat
f	Heating Rate Ratio(q/q_r)
I	Identity Matrix
J	Conditional Information Matrix (K x K)
K	Total Number of Model Parameters
$K_A, K_B \ldots$	Thermal Conductivity
L	Total Number of Spatial Node Points
P	Covariance Matrix (L x L)
Q	Model Error Covariance Matrix (L x L)
q	Heating Rate
q_r	Reference Heating Rate
RE	Freestream Reynolds Number
S	Score (Vector of Length K)
t	Time
U	Vector for Temperature Field (Vector Length L)
U_θ	Sensitivity of Temperature to Model Parameters (K Vectors of Length L)
U_∞	Free Space Temperature
X	Spatial Depth
α	Angle of Attack
β	Angle of Sideslip
δ	Deflection Angle
ε	Emissivity
θ	Model Parameters

NOMENCLATURE (CONTINUED)

λ	Dummy Variable
ρ	Density
σ	Stefan Boltzmann Constant

Superscripts

+	A Posteriori Update
^	Expected Value
-	A Priori Propagation

Subscripts

n	Number of Time Step
0	Initial Condition

ABSTRACT

A computer program named HEATEST required excessive computer time to evaluate the matrix Riccati equation for temperature covariance. Alternative numerical methods were employed to compute the Riccati equation, and the HEATEST program execution time was reduced by 70%. However, cumulative temperature covariance rose from 2.45 to 3.28 degrees. This rise was considered insignificant.

A survey was conducted of methods for computing the matrix exponential. A triangular matrix decomposition method proved to be more efficient than summing the Taylor series, especially for matrices with a large condition number. This substitution produced an overall 10% decrease in HEATEST execution time with comparable accuracy.

Simpson's Rule was used to evaluate the matrix Riccati integral term. The accuracy of this method was in the range of 5 to 9 significant digits, and computation time for the integral term was reduced by 90% for a matrix of order 13. This substitution prompted the rise in the covariance.

FORTRAN program modules and numerical examples are included.

I. INTRODUCTION

PROBLEM DESCRIPTION

This thesis attempts to solve one aspect of the problem of determining allowable limits for the reentry trajectory envelope of the Space Transportation System (STS or Space Shuttle). The Air Force Flight Test Center developed a strategy to determine this reentry envelope quantitatively. The strategy depended on the use of a computer program named HEATEST to analyze temperature data recorded during Shuttle reentry and to compute the values of certain aerothermodynamic parameters by means of an iterative estimation technique.

The HEATEST program was used successfully to process thermal data from wind tunnels, the Shuttle simulator, and a reentry maneuver during the first Shuttle test flight. The major drawback was that the program required large amounts of expensive computer processor time. According to Hodge, 1982, the Air Force Flight Test Center expended a substantial portion of a $50,000 budget on computer time to operate the HEATEST program during the first year after its development.

Computer experts at Edwards AFB used a software profiling tool to determine that most of the processing time was absorbed in performing matrix multiplications while evaluating this matrix Riccati equation:

$$P(t_n^-) = e^{A\Delta t}P(t_{n-1}^+)e^{A^T\Delta t} + e^{A\lambda}\left[\int_{t_{n-1}}^{t_n} e^{-At}Qe^{-A^Tt}dt\right]e^{A^T\lambda} \qquad (1)$$

where

A	= a tridiagonal system matrix
A^T	= the transpose of the A matrix
$\Delta t = t_n - t_{n-1}$	= time increment of heat propagation
λ	= a dummy time variable
Q	= a system error matrix
$P(t_n^-)$	= a priori value of propagated temperature covariance at time t_n
$P(t_{n-1}^-)$	= a posteriori value of covariance at time t_{n-1}

The required computation time increased rapidly as a function of the size of the matrices. For practical time limits, the size of the system matrix A was limited to order 13 (representing 13 temperature nodes), which effectively modelled only the outer inch of the Shuttle thermal protective system (TPS). Subsequent Shuttle missions will perform test maneuvers at a later time in the reentry flight profile when heat will have penetrated deeper into the TPS, so it would be desirable to process more nodes (and larger matrices) to adequately model the true Shuttle heat parameters. This problem has a particular urgency at the time of this thesis, because planned Shuttle landings at Vandenburg AFB will require trajectories with increased heat stress.

OBJECTIVES

The main objective of this project was to investigate methods for improving the performance of the HEATEST program. Specifically, the project had the following goals:

A. To make a baseline performance measurement of the numerical method originally used to calculate the exponential matrix e^{At}.

The required processor time was to be measured as a function of the order of matrix A.

B. To make a literature search of other methods for computing the exponential matrix.

C. To implement alternative numerical methods on the AFIT CYBER computer in the Fortran 5 language, and to assess their relative performance.

D. Hopefully, to discover an improved computation method, and to incorporate that technique into the HEATEST program.

E. To measure the extent of any improvement in overall HEATEST program execution time which is produced by improved computation of the matrix exponential.

F. To use standard software engineering practices when modifying the HEATEST program. These are to include modular design, top-down organization, and comprehensive documentation including source references.

The original goals were refined and extended as the research progressed. The literature search resulted in one very promising approach to the computation of the matrix exponential. The investigation of less favorable alternatives was sacrificed in order to devote full effort toward implementing a triangular matrix decomposition technique. This technique then prompted an idea for efficiently approximating the integral term of the Riccati equation by means of Simpson's rule. These numerical methods were ultimately incorporated into the HEATEST program with great success at reducing overall program execution time.

INITIAL RESEARCH STRATEGY

One possible solution might be to obtain an array processor, a computer logic unit whose architecture is optimized toward the efficient handling of matrix computations. This would not be very convenient and, more importantly, the rapid increase of processing time in response to matrix size suggests that more efficient numerical methods would be a better solution.

Equation (1) is seen to consist of two terms, each of which is computed in the HEATEST program by summing a separate Taylor series expansion. Furthermore, these series must be recomputed over many subintervals of time in order to guarantee mathematical stability and accuracy. The time consuming matrix multiplication was necessary for both Taylor series program modules, which are named MEXP and INTEG, respectively. This author chose to investigate alternative methods for computing the matrix exponential e^{at} in an attempt to produce a better MEXP routine.

The following methods were intially proposed, to be supplemented later by methods from a literature search:

A. Summing a smaller of terms in the Taylor series.

B. Taking advantage of the assumed symmetric nature of the A matrix in order to convert it to a similar diagonal form.

C. Developing a specialized matrix multiplication algorithm which is optimized toward tridiagonal matrices in order to reduce the required number of arithmetic operations.

D. Converting the A matrix to a similar triangular form, and developing a multiplication algorithm which is optimized toward that form.

These possibilities will be examined in the next two chapters of this thesis. The information in this chapter was drawn from Hodge, Phillips, and Audley, 1981 and from Hodge, 1982.

II. SURVEY OF COMPUTATION METHODS FOR THE MATRIX EXPONENTIAL

INTRODUCTION

Computation of the matrix exponential e^{At} is an important general mathematical problem with wide application to several types of physical processes. Many mathematical models involve systems of linear, constant coefficient ordinary differential equations of the form

$$x'(t) = Ax(t)$$

where A is a coefficient matrix

x(t) is a solution vector

$x(0) = x_0$ is the initial condition vector

x' represents the first derivative of x

The solution to this equation is given by

$$x(t) = e^{At}x_0.$$

This review of computation techniques effectively begins and ends with the comprehensive survey by Moler and Van Loan, 1978. That survey evaluated 19 different practical methods for computing the matrix exponential. The methods were grouped into five general classes and their relative advantages were assessed. The methods in the survey are oriented toward the solution of matrices A whose order n is less than a few hundred, so the constituent elements can be readily stored in the primary memory of contemporary computers.

Other literature sources were examined from the references quoted by Moler and Van Loan, 1978, from cross references, and from articles listed in the Government Records Annual Index (GRAI). However, many of these other papers specifically referred to the larger survey (see, for instance: Van Loan, 1977:981 and Ward, 1977:610) or else were directly referenced by Moler and Van Loan, 1978. No other article was found which contained either a comparison of methods or which had a comparable discussion of practical computer implementation considerations.

The remainder of this section will essentially consist of selected extracts from Moler and Van Loan, 1978. Unless otherwise indicated, every passage in quotation marks is from this source. The most interesting methods will be presented and critiqued as to their usefullness in solving the problem at hand.

SERIES METHODS

An algorithm can be immediately developed from the Taylor series expansion

$$e^A = I + A + A^2/2! + \ldots \tag{2}$$

One straightforward approach would be to sum the series by adding terms until the limit of machine precision is exceeded. Unfortunately, this naive approach has a serious flaw which makes it useful only for setting a lower bound on the efficiency of calculations.

The matrix exponential has a property which every algorithm must overcome. As t increases the elements of e^{At} may grow before they decay. This is true to some extent for any nonsymmetric matrix. This results in a hump which is displayed in Figure 2. An example will show why this is a problem. "Take the input

$$A = \begin{bmatrix} -49 & 24 \\ -64 & 31 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & -17 \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}^{-1} \tag{3}$$

Using a machine precision of $16^{-5} = .95 \times 10^{-6}$, the first 59 terms in the Taylor series expansion are summed to obtain the output

$$e^{A} = \begin{bmatrix} -22.25880 & -1.432766 \\ -61.49931 & -3.474280 \end{bmatrix} \tag{4}$$

Analytic calculation of

$$e^{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} e^{-1} & 0 \\ 0 & e^{-17} \end{bmatrix} \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}^{-1} \tag{5}$$

gives an output to six decimal places of

$$e^{A} = \begin{bmatrix} -0.735759 & 0.551819 \\ -1.471518 & 1.103638 \end{bmatrix} \tag{6}$$

which is not even close to the first solution. The problem is with the intermediate results $A^{16}/16!$ and $A^{17}/17!$ which have elements greater than 10^{6} in magnitude but of opposite sign. Therefore, the intermediate elements have absolute errors which are larger than the final result. It is important to realize that the problem is with arithmetic truncation and not with the series truncation."

The matrix exponential can also be calculated with a series of PADE rational approximants or continued partial fractions. Moler and Van Loan, 1978:808 state that "roundoff error makes PADE approximants unreliable." Furthermore, difficulties with matrix inversion make this method inaccurate "when the A matrix has widely spread eigenvalues." This is exactly the type of matrix that is generated by the HEATEST program.

The most sophisticated method for using a series approximation is called "scaling and squaring". This is the same time-consuming method that was originally employed in HEATEST in the module MEXP. "Roundoff error and computing costs tend to increase as $t\|A\|$, or the spread of the eigenvalues, increases." This is controlled in scaling and squaring by "exploiting the unique exponential property $e^A = (e^{A/m})^m$."

"The idea is to choose m to be a power of two for which $e^{A/m}$ can be reliably computed, and then to form $(e^{A/m})^m$ by repeated squaring. One common criterion is to choose m such that $\|A\| /m\leq 1$." The HEATEST module MEXP tries to save a step by choosing m such that $\|A\|/m\leq 3$. This technique tends to keep down the hump. "The scaling and squaring algorithm is one of the most accurate that we know."

POLYNOMIAL METHODS

The matrix exponential has many interesting polynomial decompositions. Unfortunately, they contain serious problems for computational purposes.

CAYLEY-HAMILTON THEOREM: Any function of A can be expressed in terms of a polynomial in $I, A, \ldots, A^{n-1}$. "This implies that e^{At} can be expressed as a polynomial in A with analytic coefficients in t:

$$e^{At} = \sum_{j=0}^{n-1} a_j(t)A^j \text{ "} \tag{7}$$

Moler and Van Loan, 1978:816 describe a means of generating the functions $a_j(t)$. However, this method requires a priori knowledge of the characteristic polynomial of A, is affected by roundoff error, and is structurally similar to the naive form of the Taylor series.

LAGRANGE INTERPOLATION, NEWTON INTERPOLATION, and the VANDERMONDE MATRIX: e^{At} can be expressed in terms of each of these well-known formulas. However, these mthods require a priori knowledge of the eigenvalues of A, they are "algorithms of the order of n^4 operations (prohibitive except for small n)," and they suffer problems when the eigenvalues are nearly equal (confluent).

INVERSE LAPLACE TRANSFORMS: These "inverse transforms can be expressed as a power series in t." Other techniques can also be used such as Heaviside expansion. "These methods are also $O(n^4)$ and are affected by roundoff error."

ORDINARY DIFFERENTIAL EQUATION METHODS

"Since e^{At} is a solution to ordinary differential equations, it is possible to consider methods based on numerical integration." Many powerful computer programs have been developed for solving O.D.E.'s. "Methods based on single-step formulas, multi-step formulas and variable step size" all have two features in common: they are relatively "easy to use", and they are "relatively time-consuming." "The O.D.E. programs are designed to solve a single initial value system

$$x' = f(x,t) \text{ with } x(0) = x_0 \tag{8}$$

and to obtain the solution at many values of t."

General O.D.E. solvers all suffer from "not taking advantage of the linear, constant coefficient nature" of the matrix exponential problem. Instead, they traverse "a sequence of values $0 = t_0, t_1, \ldots, t_n = t$." Moler and Van Loan, 1978:813 test three published programs that are general purpose O.D.E. solvers. The results show mainly that these programs are inconsistent and "very inefficient" for computing the matrix exponential in their present form. "They repeatedly multiply various vectors by the matrix A because, so far as they know, it may have changed since the last multiplication."

MATRIX DECOMPOSITION METHODS

Methods "based on factorizations or decompositions of the matrix A" are "likely to be the most efficient for problems involving the larger matrices and for repeated evaluations of e^{At}." If A is symmetric, then these methods are particularly simple and effective.

"All of the matrix decompositions are based on similarity transformations of the form $A = SBS^{-1}$. The power series definition of e^{At} implies $e^{At} = SE^{Bt}S^{-1}$. The idea is to find an S which is easy to compute. One difficulty is that S may be close to being singular (not invertible), which means that e^{At} may be difficult to compute accurately."

DIAGONALIZATION: "The naive approach is to take S to be the matrix whose columns are eigenvectors of A." Then we can write AV = VD where V is the matrix of column vectors

D is the matrix of diagonal eigenvalues

Then the exponential e^{Dt} of D is trivially computed by replacing each eigenvalue λ in D by $e^{\lambda t}$.

There are several problems that can arise when using this method. The first, "a theoretical difficulty, occurs when A does not have a complete set of linearly independent eigenvectors and is therefore defective. In this case, the method is unworkable because there is no invertible matrix V of eigenvectors."

The second problem, a practical difficulty, occurs when two eigenvalues are nearly equal (confluent), but not exactly so. "This can be illustrated by the example $A = \begin{bmatrix} \lambda & \alpha \\ 0 & u \end{bmatrix}$

The exponential $e^{At} = \begin{bmatrix} e^{\lambda t} & \alpha \dfrac{e^{\lambda t} - e^{ut}}{\lambda - u} \\ 0 & e^{ut} \end{bmatrix}$

where $(\lambda - u)$ is small but not negligible, computation of the divided difference

$$\frac{e^{\lambda t} - e^{ut}}{\lambda - u}$$

can result in a large relative error," especially when multiplied by large α.

A third difficulty arises from calculation of V^{-1}. This is another source of inaccuracy when using this naive method to compute the matrix exponential.

JORDAN CANONICAL FORM (JCF): "In principle, the problem caused by defective systems of eigenvectors can be solved by using a similarity transformation to the Jordan canonical form." The JCF is a special form of matrix which consists of all zero's except for square blocks of non-zero numbers along the main diagonal. These Jordan blocks reflect the vector subspaces that define the similarity class to which the matrix belongs. "The exponential of each Jordan block can be calculated in closed form, and $e^{At} = Pe^{Jt}P^{-1}$, where $J = J_1, \ldots, J_n$", a concatenation of Jordan block submatrices.

"For example,

$$\text{if } J_i = \begin{bmatrix} \lambda_i & 1 & 0 & 0 \\ 0 & \lambda_i & 1 & 0 \\ 0 & 0 & \lambda_i & 1 \\ 0 & 0 & 0 & \lambda_i \end{bmatrix} \qquad (9)$$

$$\text{then } e^{J_i t} = e^{\lambda_i t} \begin{bmatrix} 1 & t & t^2/2! & t^3/3! \\ 0 & 1 & t & t^2/2! \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (10)$$

The difficulty with this method is that it cannot be computed using floating point arithmetic. A single rounding error at the limit of machine precision may cause a multiple eigenvalue to become distinct or vice versa, which would alter the structure of J and P." For example, two matrices

$$\begin{bmatrix} \alpha & \varepsilon \\ 0 & u \end{bmatrix} \text{ and } \begin{bmatrix} \alpha & 0 \\ 0 & u \end{bmatrix}$$

will have different Jordan forms for any non-zero error ε, no matter how small.

SCHUR DECOMPOSITION: A more reliable method is to use the so-called Schur transformation $A = UTU^{-1}$, where U is a unitary (orthogonal) matrix and "T is a triangular matrix that will always exist if A has real eigenvalues. If A has complex eigenvalues, then T will be quasi-triangular with 2-by-2 blocks on the main diagonal." The $e^{At} = Ue^{Tt}U^{-1}$. This method is especially convenient because of the well-known property that the orthogonality of U implies that the inverse matrix U^{-1} is equal to the transpose matrix U^T.

The only difficulty occurs when the matrix T has eigenvalues that are nearly confluent, which can induce a magnification of roundoff error when computing e^{Tt}. "This is a general problem for all matrix decompositions of the form $A = SBS^{-1}$ and involves two objectives:

A. Make B close to diagonal so that e^{Bt} is especially easy to compute." The Jordan form epitomizes this objective.

"B. Make S well conditioned so that computational errors will not be magnified." This goal is realized by the Schur method.

A practical compromise can be achieved between these objectives. The idea is to "cluster the nearly confluent eigenvalues" into block diagonal form, using a somewhat non-orthogonal matrix S if necessary. Then special techniques can be used to compute $e^{S_i t}$ for each block. The simplest technique would be to replace each eigenvalue in the block by the average of all those in the block. The best criterion for clustering eigenvalues remains to be determined.

SPLITTING METHODS

Moler and Van Loan discuss a special characteristic of the matrix exponential and present the formula $e^A = (e^{B/m}e^{C/m})^m$. *"This approach to* computing e^A has potential interest when the exponentials e^B and e^C can be accurately and easily computed." Then it might be convenient to split $A = B + C$.

"The efficiency of this technique is somewhat difficult to assess because it depends strongly on the scalar" elements of A. Under general conditions, the splitting method is considered to be much less efficient than simply scaling and squaring the Taylor series.

OVERALL EVALUATION OF THE METHODS

A large segment of the available project time was dedicated to studying the survey by Moler and Van Loan, 1978, and a selection of the numerous papers that are referenced by them. For purposes of this thesis, the most attractive computational methods were extracted from the survey, organized as presented above, and evaluated for application to the problem at hand.

Polynomial techniques were quickly discarded because they require prior knowledge of the eigenvalues, and because they generally require $O(n^4)$ floating point operations for computer computations.

The splitting method is generally untried and does not offer any special advantage for the purposes of the HEATEST program.

Ordinary differential equation solvers seem to have potential for general application to computing functions such as the matrix exponential, and the development of an efficient software program could possibly form the basis for a dissertation. That type of investigation seems to be beyond the scope of this thesis, however. Only series and matrix decompositions remain for serious consideration.

A statement is made (Moler and Van Loan, 1978:p. 827) that the only generally competitive series method is that of scaling and squaring. This technique was the one used in the original HEATEST implementation, calculating up to 38 terms of the series for each subinterval of time for a matrix of order $n = 13$. The extensive matrix multiplications required for computing this series is the driving factor behind the excessive use of computer processor time, even though module MEXP uses the Cayley-Hamilton theorem to increase efficiency.

Matrix decomposition seemed to offer the best potential for investigation. The orthogonal property of the U transformation matrix allows the inverse U^{-1} to be easily obtained because it equals the transpose U^T, and it also preserves the condition number (see Chapter V for definition) of the matrix without magnifying the roundoff errors that are introduced by exponentiation.

A decision was made to investigate the Schur technique for matrix decomposition. At the same time, the form of the A matrix could be examined for possible ways of developing a specialized multiplication algorithm which could be used to more efficiently calculate the original series summation. This investigation and program development will be detailed in the next section.

III. COMPUTING THE MATRIX EXPONENTIAL (DEVELOPMENT OF MEXP2)

THEORY

The HEATEST program uses a module named MEXP to compute e^{At} by means of the scaling and squaring method. MEXP and its support modules are listed in Appendix A, along with a structure chart which diagrams their interaction. The first efforts to develop an improved module, MEXP2, were based on the idea of a more efficient matrix multiplication algorithm which would decrease the computation time by a linear factor. This required an examination of the characteristics that define the typical A matrix to be processed.

A random system matrix of order 10 was obtained from HEATEST and used as a test case for investigation. This matrix is displayed in Figure 3 and is seen to have the following properties:

A. Tridiagonal form.

B. Not symmetric, but nearly so.

C. Diagonally dominant. This means that each diagonal value is greater than the sum of all other numbers in that row or column.

D. All entries in the main diagonal have negative sign, while all entries in the off-diagonal rows have positive sign.

E. A wide range of values.

These properties will be seen to have great significance.

The first consideration was to attempt to exploit the inherent sparseness of the tridiagonal matrix. Several published computer codes are already available for manipulating tridiagonal matrices and for storing the non-zero values in a reduced form which saves on primary computer memory. One source is the International Mathematics and

Statistics Library (IMSL). This idea proved not to be useful for at least two reasons. First of all, the reduced form of storage would not help unless implemented throughout the entire HEATEST program, which was not feasible in the time available. More importantly, the tri-diagonal form is not closed under the operation of matrix multiplication, as shown in Figure 4. It can be seen that the resulting matrix bandwidth increases by two off-diagonal rows after each operation. This property is not conducive to a specialized algorithm for calculating series summations.

A more promising candidate would be the triangular form shown in Figure 5, which is closed under matrix multiplication. The sparseness of the triangular form can easily be exploited, as seen in Figure 7. The full formula for upper triangular multiplication is shown in Figure 6, and the resulting non-zero terms are shown in Figure 7. It would not be difficult to devise a computer code that would only include the resulting non-zero terms in its computations. The efficiency of this technique could be significant. By looking at diagonals, one can see by inspection that the number of resulting non-zero element multiplications can be described as

$$1(n) + 2(n - 1) + \ldots + n(1) = \sum_{K=1}^{n} K(n + 1 - K)$$
$$= (n + 1)\sum_{K=1}^{n} K - \sum_{K=1}^{n} K^2 \qquad (11)$$

$= (n + 1)\frac{n(n + 1)}{6} - \frac{n}{6}(2n + 1)(n + 1) = \frac{n(n + 1)}{6}[3(n + 1) - (2n + 1)$

$= \frac{n(n + 1)(n + 2)}{6}$. This sum has the value $.22(n^3)$ for $n = 10$, and it approaches the value $.16(n^3)$ as n increases. Compare this to the usual matrix multiplication which requires n^3 operations.

The algorithm above appears to be a promising technique for efficient computation of triangular series summations. A triangular scaling and squaring series method could be used to evaluate e^{Tt} for a matrix T which is obtained from the Schur decomposition of $A = UTU^{-1}$. This interesting innovation was never tested, however, due to a lack of project time and the successful implementation of an even more efficient method.

Moler and Van Loan, 1978:823, recommend using a different technique which can calculate any analytic function of a triangular matrix. This method is described in Parlett, 1976, and Fortran language program modules are available from Parlett, 1974. This method is based on a recurrence relation that exists among the elements of the triangular function matrix F, where F is obtained by using the algorithm of Equations (12) and (13).

A lemma is presented (Parlett, 1976:118) that if T is a block upper triangular matrix, then a function matrix F = f(T) is also of block upper triangular form with the same block structure. The method starts by first computing the diagonal blocks (or elements) of F from $F_{rr} = \exp(T_{rr})$. Then the upper triangular blocks (elements) can be determined from the formula

$$T_{rr}F_{rs} - F_{rs}T_{ss} = \sum_{k=0}^{s-r-1} (F_{r,r+k}\, T_{r+k,s} - T_{r,s-k}\, F_{s-k,s}) \text{ for } r<s \quad (12)$$

for successive diagonals, moving in turn away from the main diagonal. Each block F_{rs} is determined as a linear combination of lower diagonal blocks which are in row r and column s of the function matrix F. Then the right side of the equation is known, and the elements of block F_{rs}

are obtained by solving a set of simultaneous equations

$$T_{rr}F_{rs} - F_{rs}T_{ss} = R \tag{13}$$

Three cases can occur:

A. The Schur matrix is strictly triangular and the eigenvalues (diagonal elements) are all distinct. This is the simplest case, and the calculations are straightforward. A Fortran program module is supplied by Parlett, 1974:15.

B. The Schur matrix T has quasitriangular form with at least one 2-by-2 block on the main diagonal, and the eigenvalues are distinct. This case is illustrated in Figure 8. The computation of F in this case proves to be a more complicated programming exercise. The block structure must be discerned, block multiplication must be performed with variable size blocks, and the resulting linear system equation must be solved with a possible need for pivoting.

C. The Schur matrix T has confluent eigenvalues, in particular when $T_{rr} = T_{ss}$. In this situation, the basic recurrence formula breaks down and an alternative must be used. In addition, calculations in finite precision arithmetic can return inaccurate results when T_{rr} is very close, but not exactly equal, to T_{ss}. This case can be treated as follows:

1. Eigenvalues that are very close can be replaced by a confluent eigenvalue which is equal to the average of the similar eigenvalues. The menaing of "very close" is not specifically defined.

2. Confluent values are treated with an alternate formula (Parlett, 1974:3) which is derived from Newton's form of the interpolating polynomial and which is manipulated to obtain F_{rs}.

3. Then the general formula (13) can be applied. This case

requires on the order of n^4 floating point arithmetic operations. A Fortran program module is supplied for this case, also (Parlett, 1974:16).

IMPLEMENTATION

The development process was started by generating a baseline example. Under usual software development conditions, the most efficient strategy is to use the main program as the driver for new submodules. However, HEATEST was too large, in both memory and required execution time, to be operated from an interactive CYBER computer terminal with the desired immediate turnaround time. Therefore, the sample matrix in Figure 3 was obtained from HEATEST and used as a test case. A separate test program was used to compute the exponential of the test matrix by means of the original MEXP module and its supporting submodules. The result (Figure 9) was used as a baseline reference for future comparison to ensure accurate function computation. However, at this point, there was still some uncertainty as to the degree of accuracy that was returned by MEXP.

The next step was to generate the matrix exponential by using Parlett's algorithm. This required the following procedure:

A. Generate a Schur orthogonal transformation matrix U such that $A = UTU^{-1}$ where T is of upper triangular or quasitriangular form.

B. Determine the eigenvalue structure of the matrix T.

C. Apply the correct treatment of Parlett's algorithm to obtain $F = \exp(T)$.

D. Calculate $e^A = UFU^{-1}$ and print the result for comparison.

These steps will be discussed in turn.

It is recommended (Moler and Van Loan, 1978:823) to use Fortran modules ORTHES and HQR from EISPACK, 1976 to calculate the transformation matrix Q. ORTHES is used to transform any general matrix into upper Hessenburg form, in which the first subdiagonal is the only non-zero lower diagonal row. This was not required for purposes of HEATEST, since the tridiagonal form of the A matrix is effectively a subcategory of the upper Hessenburg form. The HQR module acts on the Hessenburg form to obtain an orthogonal U matrix by the iterative QR algorithm. The EISPACK HQR module was modified slightly by the author of this thesis to eliminate additional accumulations of the eigenvalues of A. This reduced the required number of input/output parameter arrays for HQR, which in turn reduced the amount of primary computer memory needed to operate the module. For the test matrix in Figure 3, HQR generated the transformation matrix U in Figure 10. It is well-known that for a given orthogonal matrix U, the inverse U^{-1} is just the transpose U^T. The matrix in Figure 10 was multiplied by its transpose to verify orthogonality and, in fact, the identity matrix was the result.

The next step in the exponentiation procedure is dependent on the eigenvalue structure of the matrix T. The test matrix in Figure 3 was transformed to the triangular matrix in Figure 11. The main diagonal elements are the eigenvalues, which are seen to be all non-zero, real, and distinct. The wide spectrum of the eigenvalues indicates a poor condition of the matrix for computational purposes.

This is all very fine for the test case, but a vital question remains as to whether every A matrix generated by HEATEST will exhibit a similar eigenvalue structure. One basis for conclusion would be to evaluate all the system matrices that are generated during one reentry flight. If these matrices all had the same structure, then an argument could be made to project that result to the general case. However, Hodge, 1982, recommended an analysis of the generating equations in the hope of mathematically proving a general result. The following paragraph presents an analysis which establishes this result, except for the confluence of eigenvalues.

HEATEST generates the A matrix according to the system of equations in Figure 12 (Hodge, Phillips, and Audley, 1981:4) which describe the energy balance of the heat propagation in the Shuttle Thermal Protective System (TPS). The first equation is of fourth order and describes the interactive heat transfer at the surface of the TPS. This equation is responsible for the non-symmetric character of the system matrices. A cursory analysis of the energy balance equations will reveal that every A matrix generated by HEATEST will be both tridiagonal and diagonally dominant. Furthermore, the elements of the main diagonal will all have the same sign, and the elements of the two off-diagonal rows will all have the same opposite sign. Diagonal dominance ensures that none of the eigenvalues will be equal to zero. Another theorem was found (Marcus and Minc, 1964:166) which proves that all Jacobi (tridiagonal) matrices with the above characteristics have a spectrum of eigenvalues which are both real and simple (no multiple values).

The above analysis is very useful, but a nebulous question remained as to whether any of the distinct eigenvalues would be so nearly confluent as to induce computational errors in the calculations. Hodge, 1982, felt that the eigenvalues would be closest to each other at the beginning of Shuttle reentry when heat transfer is slight and the surface temperatures are closest to the reference values during on-orbit conditions. This case was simulated by applying Parlett's algorithm to a similar, but worst case, test matrix which was obtained from a thin skin metal plate that generated temperature data in a wind tunnel. This metal plate had a uniform temperature through its cross-section, and the system matrix was both singular (non-invertible) and not diagonally dominant. This situation could not occur within the thermal shield itself, but it is similar to the thin skin structure of the Shuttle which is directly underneath the TPS.

The Schur transformation of this worst case matrix produced a triangular T matrix which had one eigenvalue equal to zero, but which also exhibited a spread of the eigenvalues across the spectrum. Based on this demonstrated lack of confluence, a decision was made to proceed by using the simplest case (case A) of Parlett's algorithm, which can be applied even to singular matrices as long as the eigenvalues are distinct.

The module FUNCT was developed by this author from subroutine FUNUPPD (Parlett, 1974:16). A problem was experienced with underflow when initially computing the exponential of the diagonal elements of the T matrix. This problem was solved by means of an IF statement that returns an exponential value of 0.0 whenever the input diagonal entry has a negative magnitude larger than (-43). This guarantees 14 decimal places of accuracy, which is over the limit of single precision on the CYBER computer.

A module MEXP2 was then developed by this author which used module HQR to generate the Schur transformation matrix, and then it used FUNCT to calculate F = exp(T) by Parlett's algorithm. When used on the test matrix in Figure 3, the matrix result of MEXP2 agreed exactly with Figure 9, from MEXP, to the five decimal places that were displayed.

RESULTS

Module MEXP2 was incorporated into the HEATEST program with some software difficulties that will be discussed in a later section. HEATEST was executed over a range of matrix dimensions, using both MEXP and MEXP2 for comparison as to both accuracy and execution time. Each iteration of HEATEST produced 110 executions of the exponential matrix function. These execution times were averaged and the results are presented in Table 1 and Figure 13. While returning comparable accuracy, MEXP2 showed that the Schur decomposition method was dramatically faster than the series method of module MEXP.

IV. COMPUTATION OF THE RICCATI INTEGRAL (DEVELOPMENT OF INTEG2)

THEORY

The performance of module MEXP2 demonstrated that the scaling and squaring series method was a relatively inefficient technique for computing the matrix exponential. The module INTEG used a similar series method for computing the Riccati integral, so it seemed worthwhile to investigate other techniques for computing this operation, also. Moler and Van Loan reference a paper (Levis, 1969) which analyzes the computation of this integral by the method of Simpson's rule.

Simpson's rule is an attractive numerical method when the speed of computation is important, because the rule exhibits an unusual combination of both simplicity and a high degree of accuracy. Simpson's rule approximates an integral by sections of a parabolic curve, rather than by the cruder rectangle methods. Yet, the simplest formulation of the rule consists of a sum of only three approximation terms. The accuracy of the rule is also very good. The computation error is a function of h^5, where h is the interval between approximation points. To obtain a higher degree of approximation, it would be necessary to use a more cumbersome Newton-Cotes formula with at least 5 approximation terms (Young and Gregory, 1972:369).

Simpson's rule approximates the Riccati integral

$$V = \int_{t_{n-1}}^{t_n} e^{-At} Q e^{-A't} dt \quad \text{where } A' = A \text{ transpose} \qquad \Delta t = t_n - t_{n-1} \tag{14}$$

with the formula

$$V = \frac{h}{3}\,[F(0)QF'(0) + 4F(h)QF'(h) + F(2h)QF'(2h)] \qquad (15)$$

where $h = \Delta t/2$, $F(t) = e^{-At}$

so we have

$$V = \frac{h}{3}\,[e^{-A\cdot 0}Qe^{-A'\cdot 0} + 4\,e^{-Ah}Qe^{-A'h} + e^{-A(2h)}Qe^{-A'(2h)}] \qquad (16)$$

or

$$V = \frac{t}{2\cdot 3}\,[IQI + 4e^{-At/2}Qe^{-A't/2} + e^{-At}Qe^{-At'}] \qquad (17)$$

where I = the identity matrix

Finally,

$$V = \frac{t}{6}\,[Q + 4e^{-At/2}Qe^{-A't/2} + e^{-At}Qe^{-A't}] \qquad (18)$$

Calculation of the approximation formula for the Riccati integral will require four matrix multiplications and four exponential calculations. In what ways can this situation be improved? The number of exponential computations can be cut in half by using the well-known expression $e^{-A't} = (e^{-At})'$. This equality is a consequence of the Taylor series expansion of the matrix exponential.

The necessary calculations can be further simplfied, however, by considering both terms in the Riccati equation (1). Even requiring only two exponential computations, the Simpson's rule method of computing the Riccati integral is only comparable in effort to the original Taylor series method which summed the first 12 terms in the series expansion. Consideration of the entire Riccati equation reveals that a total of three different exponential terms must be evaluated: $e^{-At/2}$, e^{-At}, and e^{At}. The use of a mathematical formula which relates these terms could lead to increased efficiency in the computation of Simpson's rule and of the entire Riccati equation.

Consider the scalar exponential relation $e^{s+t} = e^s e^t$. Does this relation also hold true for the matrix exponential? In fact, it does. A proof was found (Bellman, 1970:170) which is displayed in Figure 14. It is then possible to calculate the entire Riccati equation by making only one direct exponential evaluation, that of the term $e^{-At/2}$. Using the matrix splitting formula $e^{s+t} = e^s e^t$ yields the relation

$$e^{-At} = (e^{-At/2})(e^{-At/2}) = (e^{-At/2})^2, \qquad (19)$$

then e^{At} can be obtained by computing the direct matrix inverse $e^{At} = (e^{-At})^{-1}$. This strategy combines the qualities of easy implementation and efficient execution.

IMPLEMENTATION

The implementation strategy for incorporating INTEG2 into the HEATEST program was quite different from the case of MEXP2. MEXP2 represented a complicated arithmetic process which was performed on one piece of input data, i.e., the A matrix. A small test program was used to prove the calculation method for a particular sample input matrix, and the results could be readily verified.

INTEG2 represented a different situation. The calculation of Simpson's rule was relatively simple, but it was technically difficult to cross-check the result. The complicated structure of HEATEST made it difficult to extract matching A and Q matrices for a sample input. Therefore, the following implementation strategy was used: write a program module for INTEG2, substitute the module into HEATEST, and compare the resulting parameter estimations and error covariance to those of the previous Taylor series method.

The programming exercise actually consisted of writing two main modules: INTEG2 and TPSOSP3. Originally, TPSOSP2 constructed the entire Riccati equation by consecutively executing MEXP and INTEG to compute the first and second terms of Equation (1), respectively. The revised structure has the form of embedded shells. MEXP2 is called by INTEG2 to calculate the matrix exponential $e^{-At/2}$. INTEG2 uses this value to compute Simpson's rule in evaluating the integral function. INTEG2 is called in turn by TPSOSP3 to obtain the integral value and, also, the value of $e^{-\Lambda t}$. TPSOSP3 then uses a matrix inverse module to obtain e^{At}, generates the first term of Equation (1) from e^{At}, generates the second term of Equation (1) from the integral value, and adds the terms to obtain the final result.

Development of the modules INTEG2 and TPSOSP3 was divided into two stages. The first and most important stage was the development of software manipulation tools, such as a matrix inversion module. The second stage was the combination of these manipulation tools to construct the actual computation formula.

MEXP2 satisfied the first software tool requirement, which was to compute the matrix exponential function. The second tool requirement was for a module to perform the matrix inverse computation, which is a fairly simple process in theoretical terms. However, it is much more of a challenge to program this function in an efficient manner which does not magnify floating-point round-off errors. In this case, the Linpack User's Guide, 1979 provided efficient and reliable code for the matrix inverse function. The negative aspect of this tool was that 300 lines of code were added to HEATEST in order to perform a single operation. Evidently, such is the price of performance.

The Linpack Code was given a dry run by feeding it the orthogonal transformation matrix U which is displayed in Figure 10. In accordance with theory, the resulting inverse matrix was exactly equal to the transpose of U. This test generated a high level of confidence in the effectiveness of Linpack module SGEFA and Eispack module HQR, and also in the ability of the programmer to successfully implement these codes.

The third software tool requirement was for a module to perform the triple matrix multiplication EQE^T, where Q is presumed to be a symmetric matrix. HEATEST already contained a module named MAT4 which seemed to perform exactly this operation, so an attempt was made to interface with MAT4. MAT4 was also very attractive because it apparently worked with an exceptionally small number of intermediate storage arrays. Unfortunately, the construction of MAT4 was nearly impossible to decipher, because it used one-dimensional arrays to represent two-dimensional matrices in a rather confusing manner.

An attempt was made to validate MAT4 by incorporating it into the MEXP2 test program. As mentioned in a previous section, MEXP2 calculated the Schur transformation UTU^T. MEXP2 was reprogrammed several different ways over a period of two weeks, without achieving any success with the use of MAT4. Finally, it became necessary to trace tediously by hand the operation of MAT4 on some sample matrices of order 4-by-4. The results were illuminating.

MAT4 used minimal storage space because it only calculated the upper triangular half of the resultant matrix. When returning the final values, MAT4 simply assumed that the resulting matrix was symmetric. MAT4 had been ambiguously documented in a way that could be taken to mean that the central input matrix, rather than the output matrix, was presumed to be symmetric. Therefore, it was determined that MAT4 was useless for purposes of INTEG2, and a new module named TRI was written to satisfy this final software tool requirement, TRI was then incorporated into the MEXP2 test program as a cross-check, with perfect results.

At this point, all the necessary software tools were available and proven. The modules INTEG2 and TPSOSP3 were then written to construct the Riccati equation from the software tools, and these modules were then substituted into the HEATEST program. This second stage was performed with a certain amount of trepidation about the computation results that would be produced. The software tools had all been meticulously proven, but the final evaluation depended on combining a large number of theoretical manipulations into a couple of program module quantum jumps. It seemed unlikely that so much theory could be included in the module TPSOSP3 without an inevitable flaw, and it seemed unlikely that such a flaw could be easily isolated.

When these modules were eventually incorporated into HEATEST, it so happened that one dummy parameter was inadvertently omitted from a subroutine call. This single mistake caused several discouraging interface complications. It was necessary to expend considerable effort to merely discover the existence of a logical error. Finally, a simple debugging trace was used to locate the problem module and to correct the missing input parameter.

RESULTS

INTEG2 was successfully incorporated into the HEATEST program, and execution times were measured for INTEG versus INTEG2 over a range of matrix dimensions. The results, displayed in Table 2 and Figure 15, were as follows:

A. The time curves in Figure 15 show a significant improvement in execution speed when using Simpson's rule. It should be mentioned that the curve for INTEG2 represents only the time difference between TPSOSP2 and TPSOSP3 that resulted from the use of Simpson's rule, and does not reflect the simultaneous benefit of the improved calculation of the matrix exponential.

B. The desired aerothermodynamic parameters appear to converge to the same values whether using INTEG or INTEG2. However, the use of INTEG2 seems to require a couple of extra iterations of HEATEST to obtain the same amount of convergence as before. This is consistent with a simultaneous change from 2.45 to 3.28 in the accumulated covariance of error between predicted and measured temperature that also occurred. Some of this difference can be attributed to a magnification of roundoff error that occurred during the inversion of the matrix exponential e^{-At}, which probably destroyed about four decimal places of accuracy. The rest of the error must be taken to reflect the inherent limitations of Simpson's rule. However, the HEATEST program runs dramatically faster with module INTEG2, even when extra iterations are taken into account.

V. ANALYSIS AND CONCLUSIONS

NORMS, CONDITION NUMBER AND ACCURACY

The scaling and squaring series method, used in both modules MEXP and INTEG, is formulated on the concept of a matrix norm. A matrix norm can be defined in several different ways for theoretical purposes, but the general idea is to provide a comparative measure of the size of a matrix in terms of the magnitude of its eigenvalues. In section two of this paper, it was noted that the norm of matrix A must have a value less than 1 in order to prevent the growth of intermediate series elements. The scaling and squaring method divides A by a factor 2^k such that $\|A\|/2^k \leq 1$. The module MEXP computes the matrix exponential $e^{At/2^k}$, and the result is squared k times. Since the largest eigenvalue in a sample matrix A was found to be on the order of 28,000, it was expected that the scaling factor k would probably be tremendous because the norm would be very large. However, it was found that the module TPSOSP2 represents time in hour units, even though the actual temperature samples were taken every second in real time. Therefore, the value of t is always less than 1/3600, so the scaling effect on e^{At} was much less than expected. The module XNORM was used by MEXP and INTEG to estimate the matrix norm. The value of XNORM was examined during executions of HEATEST over a range of matrix dimensions, and the estimated norm was always in the range 75,000-77,000 with a scaling factor of $k = 5$. The norm was about half this large on all succeeding iterations of HEATEST.

Every matrix can be assigned a condition number which indicates the amplification of roundoff error that can occur when solving a linear system, such as when calculating the matrix inverse. A large condition number indicates that the linear system is very close to being linearly dependent, so the matrix is nearly singular. A similar problem can arise in the solution of matrix differential equations where the eigenvalue ratio $|e_{max}/e_{min}|$ is very large, in which case the matrix is said to be "stiff".

The A matrices of HEATEST exhibit some stiffness because the nodes of the temperature grid are necessarily closer together near the top and bottom surfaces of the Shuttle insulation. This is done in order to accurately represent the interfaces between the layers of the insulation, but the result is a matrix with a rather poor condition for computations. For example, the matrix in Figure 11 can be seen to have diagonal eigenvalues that range from 0.9 to 28,000. (For a discussion of condition numbers, refer to Young and Gregory, 1973, Vol. II: 564, 811,943)

The condition number of the HEATEST matrices was determined by calculating e_{max}/e_{min} for those eigenvalues on the main diagonal of all triangular matrices in module MEXP2. For matrices of order 13 and 30, the condition number during the initial time propagation of HEATEST was found to be in the range 75,000 to 77,000, similar to the value of the estimated norm. This would indicate an expected loss of 4 to 5 digits of precision when calculating a matrix inversion, such as the case of TPSOSP3 in obtaining e^{At} from e^{-At}.

Matrix transformations also provide an opportunity for the condition number to cause error. This particularly applies to the matrix S^{-1} which is used in the diagonal transformation $A = SDS^{-1}$. One advantage of using the Schur triangular transformation $A = UTU^{-1}$ is that all orthogonal matrices U have a condition number of 1, so no additional error is created during matrix multiplication or inversion.

The accuracy of the module HQR (that calculates the matrix U) is determined by an internal parameter named MACHEP which specifies the precision goal of the QR algorithm. The value of MACHEP was varied over the range from 10^{-5} to 10^{-13} in order to determine the effect on HEATEST. The results showed that the QR algorithm must converge very quickly, because the total execution time of HEATEST only increased from 178 seconds to 179 seconds as the precision value of MACHEP was increased. Therefore, all further use of HQR was done with the value of MACHEP set to 10^{-13}, which is the single precision limit of the CYBER computer. The values of e^{At} were then compared for matrices of order 8 after computation by MEXP and by MEXP2. The results of the two methods showed an agreement of 10 to 13 significant digits. It is difficult to judge which method is the more precise.

The computation of the Riccati integral

$$\int_{t_{n-1}}^{t_n} e^{-At} Q e^{-A't} dt \tag{19}$$

by Simpson's rule has a theoretical error bound of

$$E_n = - \frac{(\Delta t)^5}{180\ N^4} \frac{d^4}{dt^4} [e^{-At} Q e^{-A't}] \tag{20}$$

where n/2 = 1 is the number of subintervals in the approximation, according to Levin, 1969:410. Since time is represented in hour units and $\Delta t_{max} = 1/3600$, it would seem that the expected error would be close to the limit of computer precision. However, the results of modules INTEG and INTEG2 were compared for matrices of order 8, and the answers showed agreement only of 5 to 9 significant digits in the mantissa (right of the decimal point).

Evidently, the accuracy of module INTEG2 is inferior to that of INTEG, because the covariance of the a priori temperature propagation was increased slightly, from 2.45 to 3.28, for matrices of order 13. This seems to explain the observation that the aerothermodynamic parameter values appear to converge somewhat slower per iteration of HEATEST when using INTEG2. However, the parameters seem to converge to the same values whether using INTEG or INTEG2 to obtain the result. It can be seen in Figure 16 that INTEG2 is much more efficient in execution time.

EXECUTION PERFORMANCE

Analysis of the loop structure of module MEXP shows that the number of floating point operations required for each execution is

$$2n^4 + (k - 3)n^3 + (3/2)n^2 + 79n + 2k + 3$$

where n is the matrix order

k is the average scaling and squaring factor

According to Parlett, 1974:4, the operation count of module MEXP2 is only $10n^3$ to $15n^3$ for the Schur decomposition plus $(1/3)n^3$ to build up the triangular function matrix.

Analysis of the loop structure of module INTEG shows that the floating point operation count for one execution is

$$(25 + (5/2)k))n^3 + (35/2)n^2 + 16n + 45 + 2k$$

where n is the matrix order

$k = 5$ is the average scaling and squaring factor

Module INTEG2 has an operation count of only $6n^3 + 2n^2$, which consists mainly of 5 matrix multiplications and 1 matrix inversion.

The overall effect of these improvements on the HEATEST program is displayed dramatically in Figure 16. This graph shows that the large coefficients in the operation count of INTEG have a greater effect on matrices of small order than the n^4 term in the operation count of module MEXP.

SOFTWARE DESIGN CRITIQUE OF THE HEATEST PROGRAM

The most favorable design feature of the HEATEST program is the manner in which it is organized into submodules, each of which performs a well defined function. This feature was conducive to further software development because of the ease with which alternative methods could be substituted into the program.

The negative software features of HEATEST consisted of inadequate documentation, lack of source references for utility modules, and a lack of top-down logical program structure within the submodules. These features prompted some classic software development problems during the course of this project. These situations are listed as follows:

1. During the investigation of modules MEXP and INTEG it could have been instructive to print out the values of the intermediate series terms. This would have demonstrated the extent of the growth of intermediate series elements, and it might have shown whether the series could be truncated after a smaller number of terms were computed. Unfortunately, this investigation was precluded because the internal coding of these modules was almost indecipherable. MEXP, INTEG, and

several other software modules had been procured from a commercial source, and the only available documentation was limited to a description of inputs, outputs, and the general computational strategy.

This black box program construction was understandable since HEATEST was originally written as a prototype in a research environment, where internal documentation was a secondary consideration. However, this feature proved to be an obstruction to the research effort of this thesis. The replacement modules MEXP2 and INTEG2 should be a software engineering improvement to the HEATEST program because these modules have a simple sequential internal structure, all intermediate steps have descriptive comments, and source references are listed in each major submodule.

2. Documentation is almost non-existent throughout the body of the HEATEST program code, and some of the few existing comments are misleading. As a consequence, this author experienced two weeks of delay in a fruitless effort to interface INTEG2 with module MAT4. Now that HEATEST is becoming established as a useful tool, it should be upgraded with descriptive comments at the earliest opportunity. This will preclude the inevitable problems from insufficient program clarity.

3. A final example illustrates the inadequate documentation of the HEATEST program. After the development of module MEXP2, it was necessary to analyze the structure of module TPSOSP2 in order to properly interface them. Although J.K. Hodge is the person most familiar with the programming details of HEATEST, even he found it necessary to consult Audley, 1982 in order to decipher some of the mathematical functions of module TPSOSP2. At least these particular

mathematical functions are now well documented in the replacement module TPSOSP3.

SOFTWARE DEVELOPMENT PROBLEMS

As anticipated, there were two types of development problems that hindered the pace of this project. These were:

1. Programming language difficulties
2. Use of the AFIT Cyber computer resource

These problems are discussed in detail below.

It was anticipated that the Fortran 77 standard (Fortran 5) might prove to be a difficulty, since this user had last programmed in Fortran about 12 years prior to the start of this project. As it turned out, the major language problems were caused by the deficiencies of Fortran in representing variable size storage arrays. This feature caused several conflicts when the first attempts were made to interface the replacement modules into the HEATEST program. In order to overcome this difficulty, it was necessary to transmit the storage arrays throughout the entire HEATEST program by means of either COMMON statements or as dummy subroutine parameters. The danger of this strategy is that later changes to a local module can cause global errors to propagate throughout the entire program.

Another type of language problem was caused by the practical need to use HEATEST to process temperature node sets of various dimension. The HEATEST program is actually composed of several modules that are stored in special form in a Cyber UPDATE library. At the time of program compilation, there are various correction sets that must be used to provide COMMON statements that define the appropriate dimensions of the various storage arrays. This is a very flexible arrangement for the dedicated user who must process assorted data sets. However, the complexity of the correction sets and the unfamiliar UPDATE control language were impediments to this user during the software development process.

The characteristics of the AFIT Cyber computer environment caused continual frustration and difficulty throughout the entire course of this project. It will suffice merely to list the following problems that resulted from the administration of Cyber accounts and the idiosyncracies of the Cyber computer operating system:

1. Accidental revocation of the computer password for this project.

2. Periodic erasure of several working files, and the mysterious purging of one complete UPDATE library.

3. Non-availability of computer terminals, even late at night and on weekends.

4. Program turnaround times of 24 to 48 hours.

5. Occasional shutdowns of the Cyber computer system for several days, due to maintenance problems and system reorganization.

6. Unfriendly characteristics of the Cyber interactive operating system and its primitive line-oriented editor utility.

FURTHER AREAS FOR INVESTIGATION

Due to time limitations, there are several aspects of this project that could not be sufficiently examine. These topics are described here in detail.

1. Examination of the intermediate terms in the scaling and

squaring Taylor series. These terms could be checked to discover the existence of any intermediate element growth. It is also possible that the series could be further truncated, especially module MEXP which sums (25 + n) terms where n is the order of the matrix A. This could not be examined because modules MEXP and INTEG were indecipherable and would probably need to be reconstructed.

2. Moler and Van Loan, 1978:828 state that the scaling and squaring method can be implemented with the order of n^3 floating point operations. Loop analysis of module MEXP showed $O(n^4)$ operations, so this could be an inefficient implementation. An attempt could be made to rewite MEXP with $O(n^3)$ operations, and to examine the efficiency of this implementation in comparison to MEXP2, which used the Schur decomposition.

3. Use the algorithm that was developed in Section 3 of this paper for matrix multiplication of triangular matrices. This algorithm could be used in place of Parlett's method to make a series computation of the matrix function, after performing the initial Schur triangular decomposition. This method would probably be inferior to Parlett's method, however.

4. Investigate the efficiency of the diagonal transformation $A = SDS^{-1}$. This matrix decomposition is now feasible because analysis of the nature of the A matrix, from section 3 of this paper, proved that a distinct set of eigenvalues will always exist. The speed of this method would depend on the efficiency of the algorithm which constructs the transformation matrix S. This method would probably be inferior to the Schur decomposition in accuracy, however, due to the condition number of the A matrices. There would be a probable loss of 4 to 5

significant digits when performing matrix inversion to obtain the transformation matrix S^{-1}. A further loss of accuracy would occur when the matrix e^{-At} is inverted in module TPSOSP3 to obtain the value of e^{At} for calculating the first term of the Riccati equation.

5. Further analytical and computational effort could be made to investigate the accuracy of the results of Simpson's rule from module INTEG2. It is not exactly clear what effect this accuracy has on the covariance of temperature propagation in the HEATEST program. The covariance appears to be somewhat greater, but the extent of the increase is not well known. The effect of this covariance increase on the convergence of the aerothermodynamic parameter values is also in question.

CONCLUSIONS

Matrix decomposition methods can be much more efficient than the familiar Taylor series for computing the matrix exponential function, especially as the order of the matrices is increased. Many methods exist for computing the matrix exponential, but only a few are competitive in both speed and accuracy. The scaling and squaring series method is both simple and accurate, but the Schur decomposition with Parlett's method for evaluating matrix functions seems to have comparable accuracy and much greater computational efficiency.

The scaling and squaring series method can also accurately compute the Riccati integral $\int_{t_{n-1}}^{t_n} e^{-At}Qe^{-A't}dt$ but computational efficiency of this method is greatly degraded when the A matrix has a large computational condition number. In this situation, a numerical integration method such as Simpson's rule can provide a dramatic increase in the speed of execution at the cost of some loss of accuracy.

The matrix Riccati equation can be reliably computed for the HEATEST program with a considerable reduction in execution time by means of the alternative numerical methods discussed above. The covariance of temperature propagation is somewhat increased, but the slower iterative convergence that results is clearly outweighed by the increased efficiency of overall program execution.

The results of this thesis have been of immediate benefit to the U.S. Air Force. Within one week after these improved methods were perfected, the program code was transmitted to the Air Force Flight Test Center at Edwards AFB. The improved computational efficiency of the HEATEST program will not only save on budget money for computer time, but it also will allow the use of larger matrices that can model deeper heat penetration into the Shuttle thermal shield. This will result in a better definition of the envelope of reentry trajectories for the Orbiter vehicle and, therefore, an increase in its effective mission capability.

The improved HEATEST program is already in extensive use by the AFIT Aerodynamics Department, where program turnaround times have been reduced from one week to less than 24 hours. As a result, new research is being done about possible coupling effects of the aerothermodynamic parameters. The HEATEST program also has a potential for application to the reduction of wind tunnel data by the Air Force Flight Dynamics Laboratory, now that its execution cost has been substantially reduced.

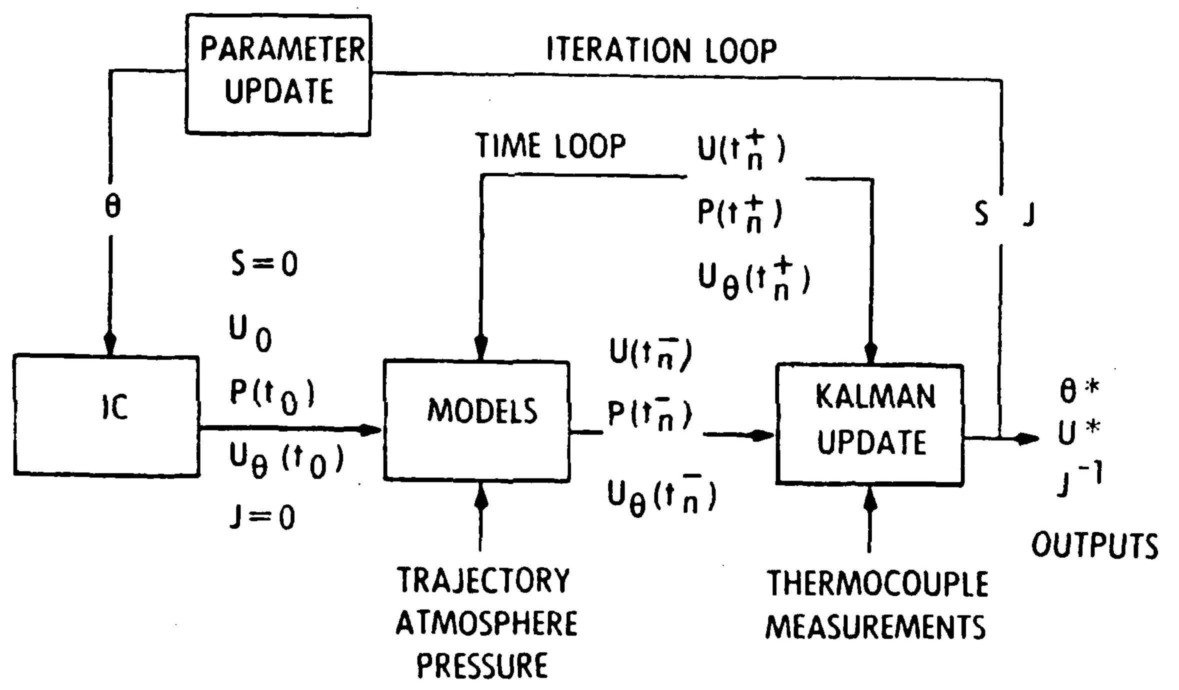


FIGURE 1. SIMPLIFIED HEATEST FLOW DIAGRAM

REFERENCE: HODGE, PHILLIPS, AND AUDLEY, 1981

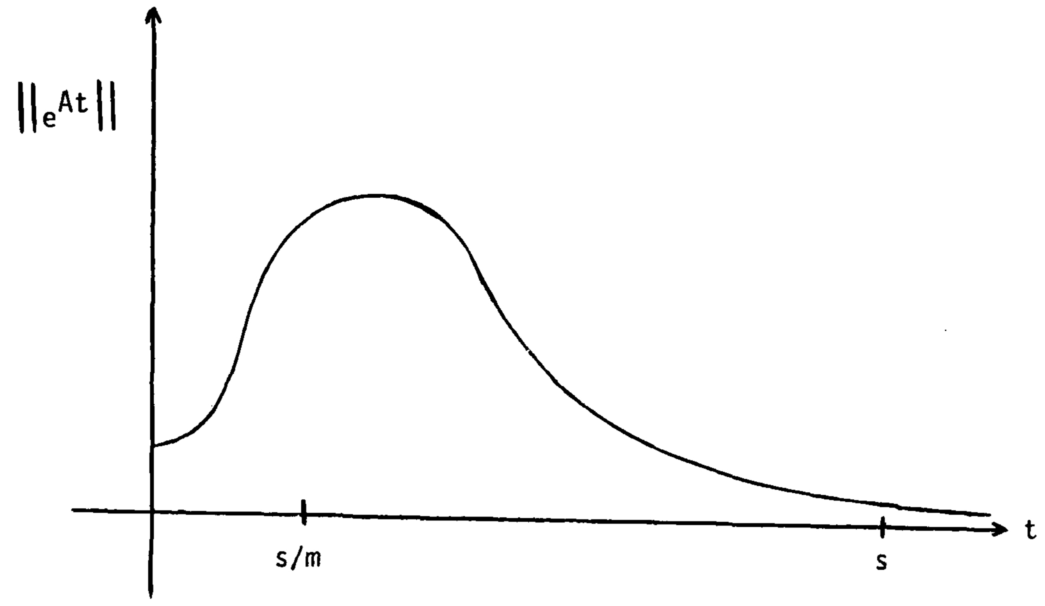


Figure 2. "The Hump"

Reference: Moler and Van Loan, 1978:804

$$\begin{bmatrix}
-18233.1 & 17795.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10069.1 & -10253.4 & 184.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 213.9 & -414.7 & 200.7 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 202.2 & -395.0 & 192.7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 90.8 & -177.9 & 87.0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 187.6 & -367.3 & 179.6 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 181.0 & -353.8 & 172.8 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 109.0 & -12368.7 & 12259.7 & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 4275.4 & -4303.4 & 23.9 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7.7 & -6.6
\end{bmatrix}$$

Figure 3. Sample System Matrix "A"

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 1 & 0 & 0 \\ 2 & 3 & 2 & 1 & 0 \\ 1 & 2 & 3 & 2 & 1 \\ 0 & 1 & 2 & 3 & 2 \\ 0 & 0 & 1 & 2 & 3 \end{bmatrix}$$

Figure 4. Tridiagonal Matrices Exhibit Bandwidth Spread Under Matrix Multiplication

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 5. Triangular Matrices Retain Their Form Under Matrix Multiplication

$$\begin{bmatrix} A_1 & A_4 & A_7 \\ A_2 & A_5 & A_8 \\ A_3 & A_6 & A_9 \end{bmatrix} \begin{bmatrix} B_1 & B_4 & B_7 \\ B_2 & B_5 & B_8 \\ B_3 & B_6 & B_9 \end{bmatrix} =$$

$$\begin{bmatrix} A_1B_1 + A_4\underline{B_2} + A_7\underline{B_3} & A_1B_4 + A_4B_5 + A_7\underline{B_6} & A_1B_7 + A_4B_8 + A_7B_9 \\ \underline{A_2}B_1 + A_5\underline{B_2} + A_8\underline{B_3} & \underline{A_2}B_4 + A_5B_5 + A_8\underline{B_6} & \underline{A_2}B_7 + A_5B_8 + A_8B_9 \\ \underline{A_3}B_1 + \underline{A_6B_2} + A_9\underline{B_3} & \underline{A_3}B_4 + \underline{A_6}B_5 + A_9\underline{B_6} & \underline{A_3}B_7 + \underline{A_6}B_8 + A_9B_9 \end{bmatrix}$$

Figure 6. Matrix Multiplication by Elements
Lower Triangular Elements are Underlined.

$$\begin{bmatrix} A_1B_1 & A_1B_4 + A_4B_5 & A_1B_7 + A_4B_8 + A_7B_9 \\ 0 & A_5B_5 & A_5B_8 + A_8B_9 \\ 0 & 0 & A_9B_9 \end{bmatrix}$$

Figure 7. Remaining Terms When Lower Triangular
Elements are all Zero

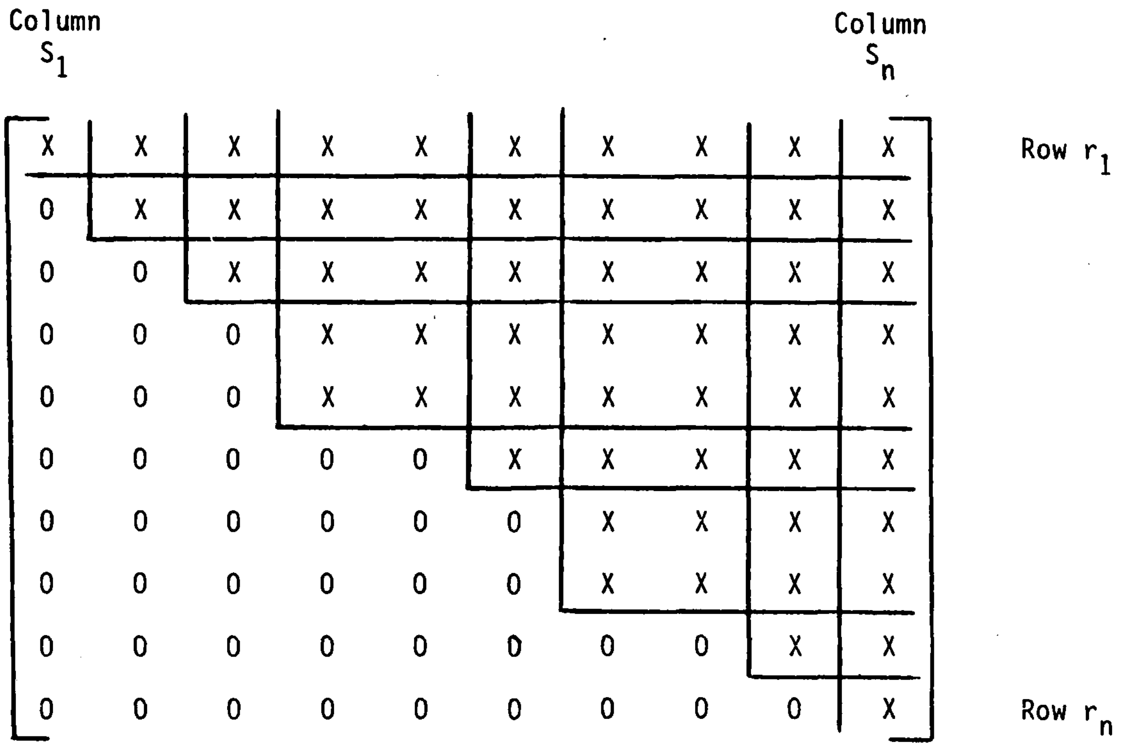


Figure 8. Shows a Quasi-Triangular Matrix, and the Resulting Blocks of Various Sizes That Must be Multiplied for Parlett's Algorithm

$$\begin{bmatrix}
.00008 & .00015 & .00029 & .00046 & .00135 & .00080 & .00098 & .00183 & .00526 & .02226 \\
.00008 & .00015 & .00030 & .00047 & .00139 & .00082 & .00100 & .00188 & .00539 & .02280 \\
.00019 & .00034 & .00068 & .00108 & .00319 & .00190 & .00230 & .00433 & .01242 & .05252 \\
.00030 & .00055 & .00109 & .00173 & .00510 & .00303 & .00368 & .00691 & .01985 & .08394 \\
.00042 & .00076 & .00151 & .00240 & .00708 & .00421 & .00511 & .00959 & .02755 & .11648 \\
.00054 & .00098 & .00194 & .00308 & .00906 & .00539 & .00654 & .01229 & .03528 & .14918 \\
.00066 & .00120 & .00237 & .00377 & .01109 & .00659 & .00800 & .01504 & .04318 & .18255 \\
.00078 & .00142 & .00282 & .00447 & .01315 & .00782 & .00949 & .01783 & .05121 & .21651 \\
.00079 & .00142 & .00282 & .00447 & .01317 & .00783 & .00950 & .01786 & .05128 & .21680 \\
.00107 & .00194 & .00384 & .00610 & .01795 & .01066 & .01295 & .02433 & .06986 & .29536
\end{bmatrix}$$

Figure 9. The Exponential of Matrix "A" in Figure 3

$$\begin{bmatrix}
.87222 & .00000 & -.14258 & .04958 & .30137 & .17914 & -.29466 & .07302 & -.03673 & .00350 \\
-.48909 & .00000 & -.24893 & .08706 & .53498 & .32024 & -.52975 & .13193 & -.06644 & .00634 \\
.00377 & .00000 & .69036 & -.17748 & -.32708 & .09861 & -.55613 & .22474 & -.12346 & .01257 \\
-.00003 & .00000 & -.62622 & .04816 & -.60030 & -.16822 & -.30876 & .29696 & -.18150 & .01995 \\
.00000 & .00000 & .14352 & .14996 & .29339 & -.33992 & .27518 & .66777 & -.47980 & .05873 \\
.00000 & -.00011 & -.14426 & -.72138 & -.04469 & .52753 & .29982 & .19468 & -.22251 & .03490 \\
.00000 & .01004 & .08671 & .64130 & -.25808 & .64699 & .23018 & .03841 & -.19915 & .04237 \\
.00000 & -.94469 & -.00082 & -.00840 & .00763 & -.03386 & -.02947 & -.19239 & -.24913 & .07962 \\
.00000 & .32781 & -.00507 & -.04408 & .02988 & -.11721 & -.09188 & -.55549 & -.71181 & .22862 \\
.00000 & -.00015 & .00005 & .00050 & -.00055 & .00319 & .00467 & .08776 & .24071 & .96660
\end{bmatrix}$$

Figure 10. Orthogonal Schur Transformation Matrix "U" for the "A" Matrix in Figure 3

$$\begin{bmatrix}
-28211.8 & 0.0 & -2206.3 & 771.4 & 4739.4 & 2836.5 & -4691.7 & 1168.2 & -588.3 & 56.1 \\
0.0 & -16624.1 & 45.6 & 393.0 & -260.8 & 1011.6 & 783.9 & 4696.0 & 6010.1 & -1925.0 \\
0.0 & 0.0 & -662.0 & -2.0 & -13.7 & 20.0 & -19.3 & -65.6 & 37.1 & -3.3 \\
0.0 & 0.0 & 0.0 & -565.7 & -9.3 & 13.8 & -23.8 & -90.5 & -12.3 & 10.5 \\
0.0 & 0.0 & 0.0 & 0.0 & -365.6 & 13.2 & -17.2 & -42.8 & 63.6 & -11.3 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -242.1 & 5.8 & 3.3 & -83.0 & 18.2 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -136.3 & -27.7 & -7.7 & 2.7 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -45.7 & -12.3 & -7.1 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -19.7 & -14.3 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.9
\end{bmatrix}$$

Figure 11. Triangular Exponential Matrix "T" from Matrix "A" in Figure 3

(A) $$C_A \rho_A \Delta X_A / 2 \dot{U}_1 = - K_A/\Delta X_A U_1 + K_A/\Delta X_A U_2 - \sigma\varepsilon\,(U_1^4 - U_\infty^4) + f(\alpha,\beta,\delta,RE) q_r$$

(B) $$(C_A \rho_A \Delta X_A/2 + C_B \rho_B \Delta X_2/2)\dot{U}_2 = K_A/\Delta X_A U_1 - (K_A/\Delta X_A + K_B/\Delta X_2) U_2 + K_B/\Delta X_2 U_3$$

(C) $$C_B \rho_B (\Delta X_{i-1} + \Delta X_i)/2 \dot{U}_i = K_B/\Delta X_{i-1}\, U_{i-1} - (K_B/\Delta X_{i-1} + K_B/\Delta X_i) U_i + K_B/\Delta X_i\, U_{i+1}$$

(D) $$(C_B \rho_B \Delta X_{L-3}/2 + C_C \rho_C \Delta X_C/2)\dot{U}_{L-2} = K_B/\Delta X_{L-3} - (K_B/\Delta X_{L-3} + K_C/\Delta X_C) U_{L-2} + K_C/\Delta X_C\, U_{L-1}$$

(E) $$(C_C \rho_C \Delta X_C/2 + C_D \rho_D \Delta X_D/2)\dot{U}_{L-1} = K_C/\Delta X_C U_{L-2} - (K_C/\Delta X_C + K_D/\Delta X_D) U_{L-1} + K_D/\Delta X_D U_L$$

(F) $$C_D \rho_D \Delta X_D / 2 \dot{U}_L = K_D/\Delta X_D U_{L-1} - K_D/\Delta X_D U_L$$

Figure 12. System Equations Which Generate Matrix "A"

Table I

Comparison Data for Modules MEXP and MEXP2

Matrix Size	Execution Time (Seconds)	
	MEXP	MEXP2
5 x 5	.014	.012
8 x 8	.071	.037
9 x 9	.110	—
10 x 10	.163	.064
11 x 11	.232	—
12 x 12	.327	—
13 x 13	.435	.119
14 x 14	.595	—
15 x 15	.778	.198
16 x 16	1.016	—
17 x 17	1.267	—
20 x 20	—	.396
25 x 25	—	.718
30 x 30	—	1.142

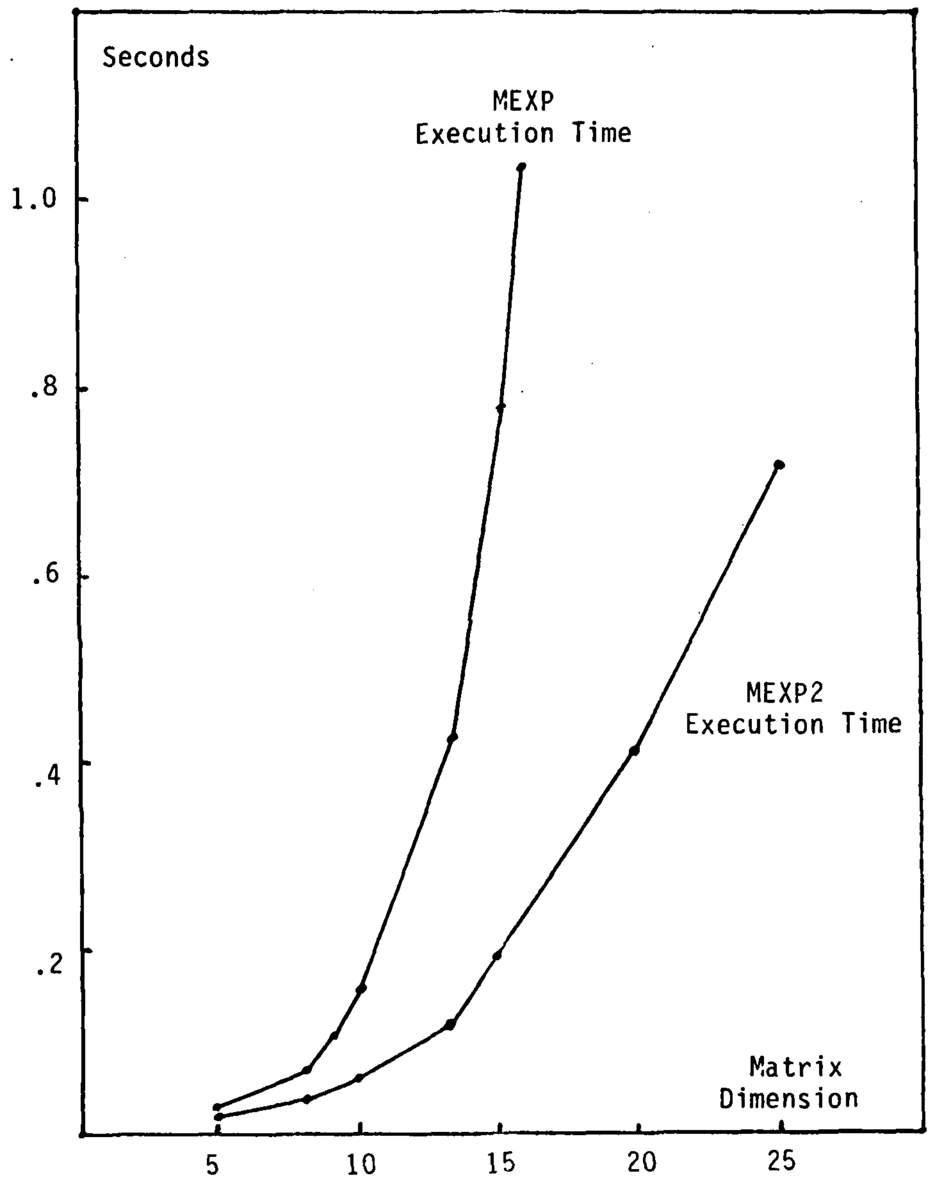


Figure 13. Matrix Exponential Calculation Times

$$e^{A(s + t)} = e^{As}e^{At}$$

Proof:

Using the series expansions for the three exponentials and the fact that absolutely convergent series may be rearranged in arbitrary fashion, we have

$$e^{As}e^{At} = \left[\sum_{k=0}^{\infty} \frac{A^k s^k}{k!}\right]\left[\sum_{m=0}^{\infty} \frac{A^m t^m}{m!}\right]$$

$$= \sum_{n=0}^{\infty} A^n \left[\sum_{k+m=n} \frac{s^k t^m}{k!m!}\right]$$

$$= \sum_{n=0}^{\infty} A^n \frac{(s + t)^n}{n!}$$

$$= e^{A(s+t)}$$

Ref: Bellman, 1970: p. 170

Figure 14. Proof of Matrix Exponential Theorem

Table II

Comparison Data for Modules INTEG and INTEG2

Matrix Size	Execution Time (Seconds)	
	INTEG	INTEG2
8 x 8	.170	.020
10 x 10	.314	.033
13 x 13	.649	.062
15 x 15	.981	.088
20 x 20	2.233	.175
25 x 25	4.251	.311
30 x 30	7.272	.509

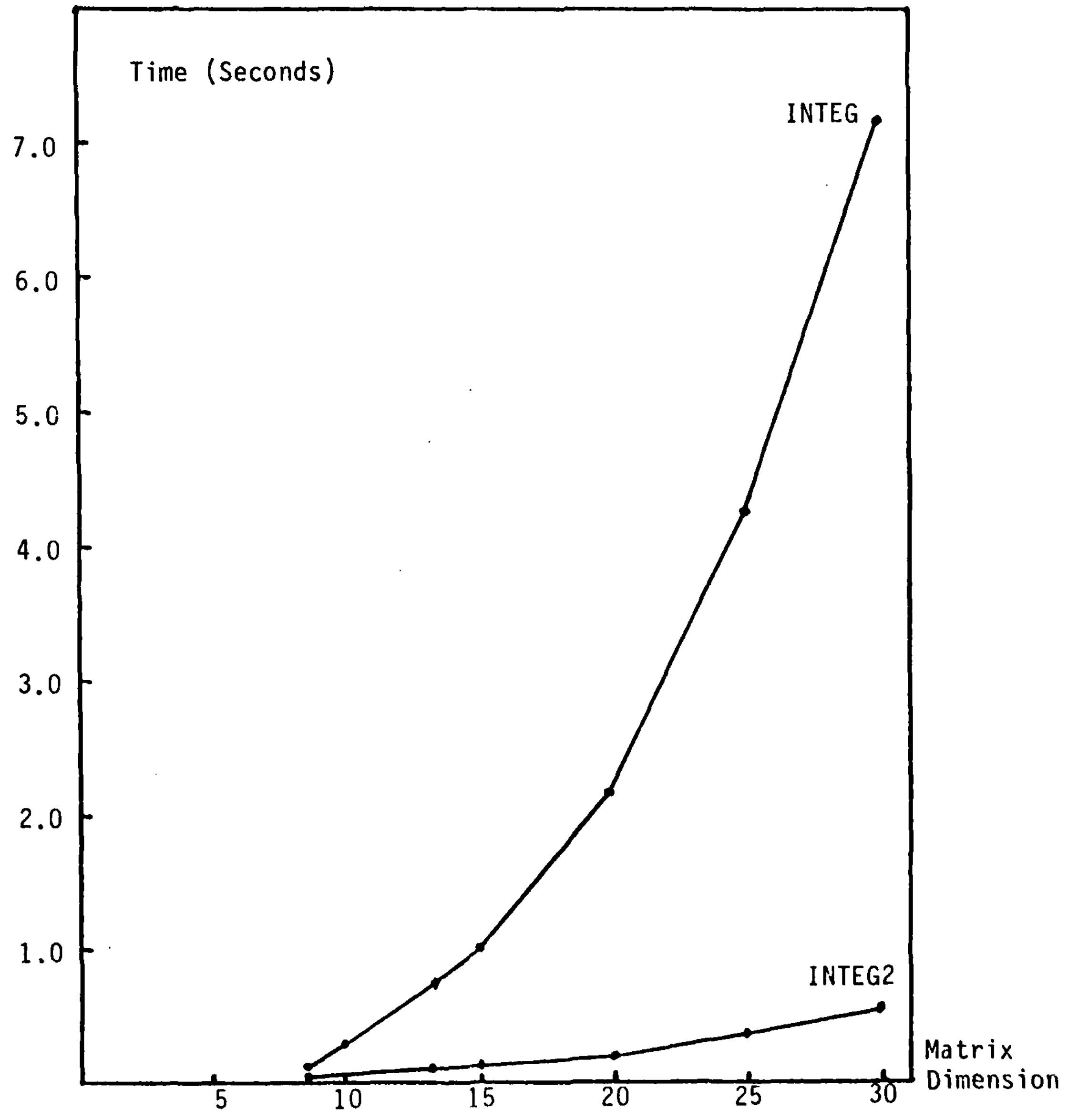


Figure 15. Execution Time of the Riccati Integral

Table III

Total Execution Time of the Heatest Program Using Module Substitutions

Execution Time Includes Initial Time Propagation Plus One Parameter Iteration

Matrix Size	Execution Time (Seconds)		
	MEXP, INTEG	MEXP2, INTEG	MEXP2, INTEG2
8 x 8	74.5	68.6	38
10 x 10	116.3	109.2	50
13 x 13	266.8	198.3	76
15 x 15	413.5	290.6	103
20 x 20	—	—	190
23 x 23	—	—	252
24 x 24	—	—	287
26 x 26	—	—	352

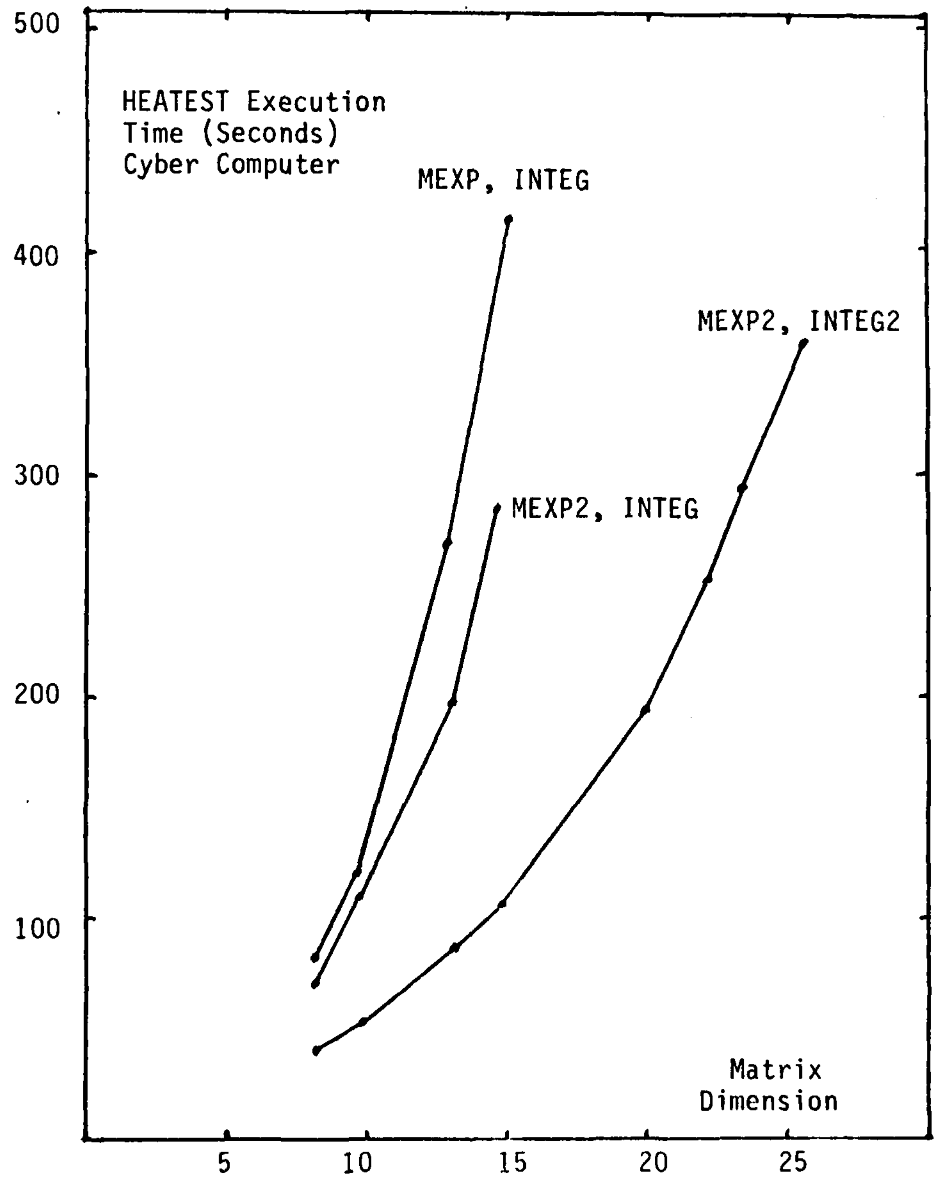


Figure 16. Total Execution Time of HEATEST Program

BIBLIOGRAPHY

1. Audley, David R., Private Communications, Air Force Institute of Technology, 1982.

2. Bellman, Richard, Introduction to Matrix Analysis (Second Edition), New York: McGraw-Hill Book Company, 1970.

3. Dongarra, J.J., J.R. Bunch, C.B. Moler, and G.W. Stewart, Linpack User's Guide, Philadelphia: Society for Applied and Industrial Mathematics, 1979.

4. Hodge, James K., Private Communications, Air Force Institute of Technology, 1982.

5. -----, P.W. Phillips and D.R. Audley, "Flight Testing a Manned Lifting Reentry Vehicle (Space Shuttle) for Aerothermodynamic Performance", Paper #AIAA-81-2421, Proceedings of the First AIAA Flight Test Conference, Las Vegas, Nevada, November 1981.

6. IMSL, Inc., 7500 Bellaire Blvd., Houston, Texas 77036.

7. Levis, A.H., "Some Computational Aspects of the Matrix Exponential", IEEE Transactions of Automatic Control, 14:410-411, (1969).

8. Marcus, Marvin and Henryk Minc., A Survey of Matrix Theory and Matrix Inequalities, Boston: Allyn and Bacon, Inc., 1964.

9. Moler, Cleve and Charles Van Loan, "Nineteen Dubious Ways to Compute the Exponential of a Matrix", SIAM Review, 20:801-836, (October 1978).

10. Parlett, B.N., Computation of Functions of Triangular Matrices, Memo #ERL-M481, University of California at Berkeley, November 1974, (AD-A 005 916).

11. ----- "A Recurrence Among the Elements of Functions of Triangular Matrices", Linear Algebra Appl, 14:117-121, (1976).

12. Smith, B.J., J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema and C.B. Moler, Matrix Eigensystem Routines, EISPACK Guide (Second Edition), New York: Springer-Verlag Book Company, 1976.

13. Van Loan, Charles F., "The Sensitivity of the Matrix Exponential", SIAM J. Numer. Anal., 14:971-981, (1977).

14. Ward, R.C., "Numerical Computation of the Matrix Exponential with Accuracy Estimate", SIAM J. Numer. Anal., 14:600-610, (1977).

15. Young, David M. and Robert T. Gregory, A Survey of Numerical Mathematics, Reading, Massachusetts, Addison-Wesley Publishing Company, 1973.

APPENDIX A

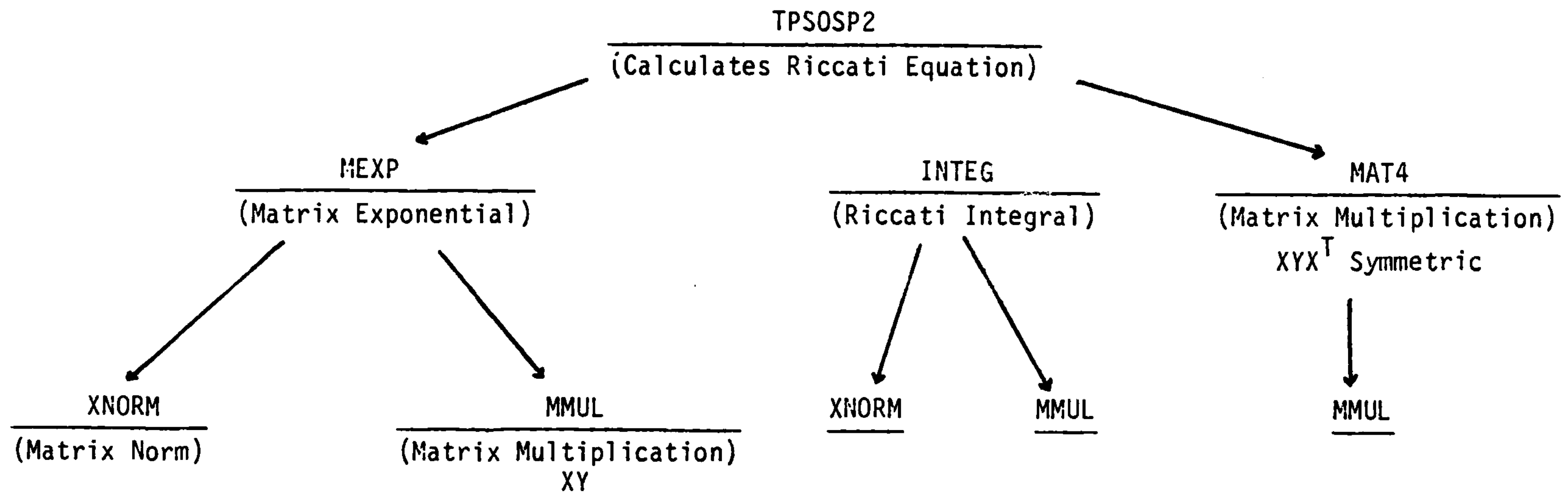


Figure 17. Structure Chart of Original Program Modules

```
      SUBROUTINE TPSOSP2 (DT)
      DIMENSION PHI(8,8),PC(8,8),QD(8,8),QDT(8,8)
      DIMENSION QUE(8,8),A(8,8)
      COMMON/MAIN/NPTPC,NPTSS,PHI,PC,QD,QDT,QUE,A
      IF(DT .LE. 0.0) GO TO 999
      DELT=DT/3600.
C
C  COMPUTE LINEARIZED STATE TRANSITION MATRIX, PHI(K,K+1)
C
      NPTSS=NPTPC
      CALL MEXP(NPTSS,A(1,1),DELT,PHI(1,1)
C
COMPUTE MODEL NOISE COVARIANCE, QD(K+1)
C
      CALL EQUATE(NPTSS,NPTSS,QD(1,1),QUE(1,1)
      CALL MSCALE(NPTSS,NPTSS,A(1,1),-1.0,A(1,1)
      CALL INTEG(NPTSS,A(1,1),QD(1,1),QDT(1,1),DELT)
      CALL MAT4(NPTSS,NPTSS,QDT(1,1),PHI(1,1),QD(1,1))
      CALL MSCALE(NPTSS,NPTSS,A(1,1),-1.0,A(1,1))
C
C  COMPUTE ONE-STEP PREDICTED APRIORI COVARIANCE, PC(K+1)
C
      CALL MAT4(NPTSS,NPTSS,PC(1,1),PHI(1,1),QDT(1,1))
      DO 520  IPC=1,NPTSS
      DO 520  JPC=1,NPTSS
  520 PC(IPC,JPC) = QDT(IPC,JPC)+QD(IPC,JPC)
      NPTSS=INF
  999 RETURN
      END
..
```

```
      SUBROUTINE MEXP(N,A,T,EA)
      DIMENSION A(1),EA(1),C(30),D(31),E(30)
      COMMON/MAIN1/NDIM,X(1)
      NDIM=NDIM+1
      NN=NDIM*N
      NM1=N-1
      IF(N .GT. 1) GO TO 5
      EA(1)=EXP(T*A(1))
      RETURN
    5 W=1.0
      DO 10  I=1,NN,NDIM
      IL=I+NM1
      DO 10  J=I,IL
   10 EA(J)=A(J)
      C1=XNORM(N,A)
      IND=0
      L=1
      T1=T
   15 IF(ABS(T1*C1).LE. 3.0) GO TO 20
      T1=T1/2.
      IND=IND+1
      GO TO 15
   20 C2=0.
      DO 25  I=1,NN,NDIM1
   25 C2=C2-EA(I)
      C2=C2/FLOAT(L)
      C(L)=C2
      D(L+1)=0.
      II=N+1-L
      E(II)=W
      II=1
      DO 35  I=1,NN,NDIM
      IL=I+NM1
      DO 30  J=I,IL
   30 X(J)=EA(J)
      X(II)=X(II)+C2
   35 II=II+NDIM1
      IF(L .EQ. N) GO TO 40
      CALL MMUL(X,A,N,N,N,EA)
      W=W*T1/FLOAT(L)
      L=L+1
      GO TO 20
   40 CONTINUE
```

```
C        CAN CHECK X:0 FOR ACCURACY
      J=N+25
      DO 50  L=N,J
      DO 45  K=1,N
      D(K)=D(K+1)-W*C(K))*T1/FLOAT(L)
   45 E(K)=E(K)+D(K)
   50 W=D(1)
      II=1
      DO 60  I=1,NN,NDIM
      IL=I+NM1
      DO 55  J=I,IL
   55 EA(J)=E(1)*A(J)
      EA(II)=EA(II)+E(2)
   60 II=II+NDIM1
      IF(N .EQ. 2) GO TO 85
      DO 80  L=3,N
      CALL MMUL(EA,A,N,N,N,X)
      II=1
      C2=E(L)
      DO 75  I=1,NN,NDIM
      IL=I+NM1
      DO 70  J=I,IL
   70 EA(J)=X(J)
      EA(II)=EA(II)+C2
   75 II=II+NDIM1
   80 CONTINUE
  85 IF(IND .EQ. 0) RETURN
      DO 100  L=1,IND
      DO 90  I=1,NN,NDIM
      IL=I+NM1
      DO 90  J=I,IL
   90 X(J)=EA(J)
  100 CALL MMUL(X,X,N,N,N,EA)
      RETURN
      END
      SUBROUTINE INTEG(N,A,C,S,T)
C     S=INTEGRAL EA*C*EA' FROM 0 TO T
C     C IS DESTROYED
      DIMENSION A(1),C(1),S(1),COEF(15)
      COMMON/MAIN1/NDIM,X(1)
      NDIM1=NDIM+1
      NN=N*NDIM
      NM1=N-1
      NT=13
      NTM1=NT-1
      IND=0
      ANORM=XNORM(N,A)
      DT=T
    5 IF(ANORM*ABS(DT) .LE. 0.5) GO TO 10
      DT=DT/2.
      IND=IND+1
      GO TO 5
   10 DO 15  I=1,NN,NDIM
      J=I+NM1
      DO 15  JJ=I,J
   15 S(JJ)=DT*C(JJ)
      T1=DT**2/2.
      DO 25  IT=3,17
      CALL MMUL(A,C,N,N,N,X)
      DO 20  I=1,N
      II=(I-1)*NDIM
      DO 20  JJ=I,NN,NDIM
      II=II+1
      C(JJ)=(X(JJ)+X(II))*T1
   20 S(JJ)=S(JJ)+C(JJ)
   25 T1=DT/FLOAT(IT)
      IF(IND .EQ. 0) GO TO 100
      COEF(NT)=1.0
      DO 30  I=1,NTM1
      II=NT-I
   30 COEF(II)=DT*COEF(II+1)/FLOAT(I)
      DO 40  I=1,NN,NDIM
      II=1
      J=I+NM1
      DO 35  JJ=I,J
   35 X(JJ)=A(JJ)*COEF(1)
```

```
      X(II)=X(II)+COEF(2)
   40 II=II+NDIM1
      DO 55 L=3,NT
      CALL MMUL(A,X,N,N,N,C)
      II=1
      T1=COEF(L)
      DO 55  I=1,NN,NDIM
      J=I+NM1
      DO 50  JJ=I,J
   50 X(JJ)=C(JJ)
      X(II)=X(II)+T1
   55 II=II+NDIM1
C     X=EXP(A*DT)
      L=0
   60 L=L+1
      CALL MMUL(X,S,N,N,N,C)
      II=1
      DO 90  I=1,N
      J=II
      IF(I .EQ. 1) GO TO 75
      DO 70  JJ=I,II,NDIM
      S(JJ)=S(J)
   70 J=J+1
   75 DO 85  JJ=1,N
      KK=JJ
      DO 80  K=1,NN,NDIM
      S(J)=S(J)+C(K)*X(KK)
   80 KK=KK+NDIM
   85 J=J+NDIM
      DO 87  JJ=I,NN,NDIM
   87 C(JJ)=X(JJ)
   90 II=II+NDIM
      IF(L .EQ. IND) GO TO 100
      CALL MMUL(C,C,N,N,N,X)
      GO TO 60
  100 CONTINUE
      RETURN
      END
      SUBROUTINE MMUL(X,Y,N1,N2,N3,Z)
      DIMENSION X(1),Y(1),Z(1)
      COMMON/MAIN1/NDIM
      NEND3=NDIM*N3
      NEND2=NDIM*N2
      DO 1  I=1,N1
      DO 1  J=I,NEND3,NDIM
      TM= 0.
      K=I
      KK=J-1
    5 KK=KK+1
      TM=TM+X(K)*Y(KK)
      K=K+NDIM
      IF(K .LE. NEND2) GO TO 5
    1 Z(J)=TM
      RETURN
      END
..
```

```
      FUNCTION XNORM(N,A)
C   COMPUTES AN APPROXIMATION TO NORM OF A. NOT A BOUND.
      DIMENSION A(1)
      COMMON/MAIN1/NDIM
      NDIM1=NDIM+1
      NN=N*NDIM
      C1=0.
      TR=A(1)
      IF(N .EQ. 1) GO TO 20
      I=2
      DO 10  II=NDIM1,NN,NDIM
      J=II
      DO 5  JJ=I,II,NDIM
      C1=C1+ABS(A(J)*A(JJ))
    5 J=J+1
      TR=TR+A(J)
   10 I=I+1
      TR=TR/FLOAT(N)
      DO 15  II=1,NN,NDIM1
   15 C1=C1+(A(II)-TR)**2
   20 XNORM=ABS(TR)+SQRT(C1)
      RETURN
      END
..
```

```
      SUBROUTINE MAT4(N1,N2,XYZ)
C     Z=YXY" X=X" IS N2XN2, Y IS N1XN2, Z IS N1XN1
      DIMENSION X(1),Y(1),Z(1)
      COMMON/MAIN1/NDIM
      CALL MMUL(Y,X,N1,N2,N2,Z)
      NN2=N2*NDIM
      DO 3  I=1,N1
      IM1= I-1
      II=IM1*NDIM
      JJ=I+II
      DO 2  J=I,N1
      TEMP= 0.
      KK=J
      DO 1 K=I,NN2,NDIM
      TEMP=TEMP+Y(K)*Z(K,K)
    1 KK=KK+NDIM
      Z(JJ)=TEMP
    2 JJ=JJ+NDIM
      JJ=I
      K=II+1
      KK=II+IM1
      DO 3  J=K,KK
      Z(JJ)=Z(J)
      JJ=JJ+NDIM
    3 CONTINUE
      RETURN
      END
```

APPENDIX B

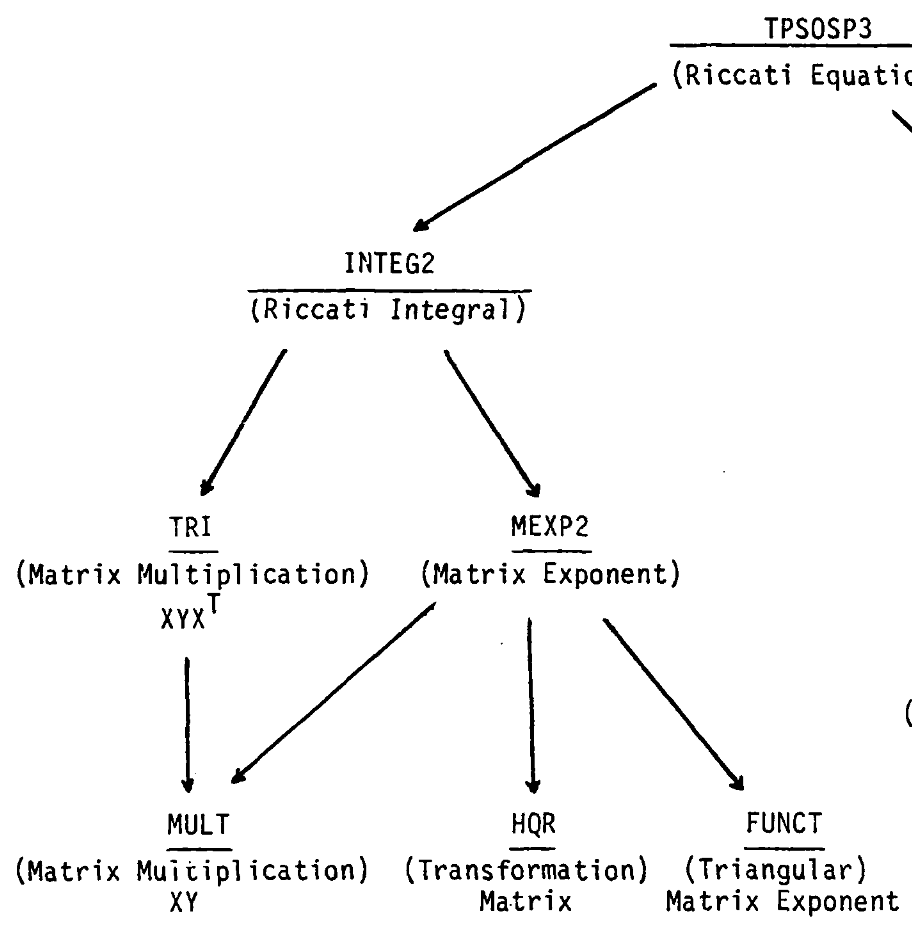


Figure 18. Structure Chart of New Program Modules

```
*DECK FTPSOSP3
      SUBROUTINE TPSOSP2(DT)
C   FUNCTION- COMPUTES THE MATRIX RICCATI EQUATION
C      PC=(EXP(A*T*PC*EXP(A'*T)) +
C         + EXP(A*T)*INTEGRAL(PHI DT)*EXP(A'*T)
C      WHERE PHI = EXP(A*T)*QUE*EXP(-A'*T)
C        AND A' = TRANSPOSE OF MATRIX A
C   INPUT VALUES:
C      A: SYSTEM SOLUTION MATRIX.
C      QUE: MODEL ERROR MATRIX.
C      PC: VALUE OF COVARIANCE PROPAGATED OVER TIME.
C      DT: TIME INCREMENT IN UNITS OF SECONDS.
C      NPTSS: GLOBAL LEADING DIMENSION OF ALL ARRAYS.
C      NPTPC: THE ORDER OF SUBMATRICES BEING MANIPULATED.
C      QD,QDT,PHI: STORAGE ARRAYS. WILL BE DESTROYED.
C   OUTPUT VALUES:
C      PC: VALUE OF A PRIORI COVARIANCE AFTER PROPAGATION.
C      NPTPC,NPTSS,DT,QUE: INPUT VALUES ARE UNCHANGED.
C      A,QD,QDT,PHI: INPUT VALUES ARE DESTROYED.
      DIMENSION IPVT(8),WORK(8)
      DIMENSION PHI(8,8),PC(8,8),QD(8,8),QDT(8,8)
      DIMENSION QUE(8,8),A(8,8)
      COMMON/MAIN/NPTPC,NPTSS,PHI,PC,QD,QDT,QUE,A
      IF(DT .LE. 0.) GO TO 999
      T=DT/3600
C   TIME T IS NOW IN HOUR UNITS.
      CALL INTEG(NPTPC,A,T,QUE,PHI,QD,QDT,NPTSS)
C   NOW QDT = INTEGRAL(PHI DT)
C        A = EXP(-A*T)
      CALL SGEFA(A(1,1),NPTSS,NPTPC,IPVT(1))
      CALL SGEDI(A(1,1),NPTSS,NPTPC,IPVT(1),WORK(1))
C   NOW A = EXP(A*T) AFTER MATRIX INVERSION.
      CALL MAT4(NPTPC,NPTPC,QDT(1,1),A(1,1),QD(1,1))
C   NOW QD = EXP(A*T)*INTEGRAL(PHI DT)*EXP(A'*T)
      CALL MAT4(NPTPC,NPTPC,PC(1,1),A(1,1),QDT(1,1))
C   NOW QDT = EXP(A*T)*PC*EXP(A'*T)
      DO 20  I=1,NPTPC
      DO 20  J=1,NPTPC
   20 PC(I,J)= QDT(I,J)+ QD(I,J)
C   NOW PC HOLDS THE SUM OF BOTH TERMS IN
C    THE RICCATI EQUATION.
  999 RETURN
      END
..
```

```
*DECK MEXP2
      SUBROUTINE MEXP(N,SUB1,TIME,SUB2,Q,QT,N2)
C   FUNCTION- COMPUTES MATRIX EXPONENTIAL EXP(SUB1*TIME)
C      BY SCHUR METHOD OF TRIANGULAR DECOMPOSITION.
C   INPUT VALUES:
C      SUB1: MATRIX TO BE EXPONENTIATED.
C      TIME: TIME INCREMENT IN AR ITRARY UNITS.
C      SUB2: STORAGE ARRAY. CONTENTS WILL BE DESTROYED.
C      Q,QT: STORAGE ARRAYS.CONTENTS WILL BE DESTROYED.
C      N2:   THE GLOBAL LEADING DIMENSION OF ALL ARRAYS.
C      N:    THE ORDER OF SUBMATRICES TO BE MANIPULATED.
C   OUTPUT VALUES:
C      SUB2: HOLDS THE MAIN RESULT EXP(SUB1*TIME).
C      SUB1,Q,QT: INPUT CONTENTS HAVE BEEN DESTROYED.
C      TIME,N,N2: INPUT VALUES ARE UNCHANGED.
      DIMENSION SUB1(N2,N2),SUB2(N2,N2)
      DIMENSION Q(N2,N2),QT(N2,N2)
C   MULTIPLY ELEMENTS OF SUB1 BY TIME
      DO 102  I=1,N
      DO 102  J=1,N
      SUB1(I,J)=SUB1(I,J)*TIME
  102 SUB2(I,J)=SUB1(I,J)
C   GENERATE IDENTITY MATRIX FOR INPUT Q, FOR HQR
      DO 30  I=1,N
      DO 30  J=1,N
      Q(I,J)=0.
   30 IF(I .EQ. J) Q(I,J)=1.
      CALL HQR(N,SUB2,Q,N2)
C   MATRIX SUB2 HAS BEEN DESTOYED
C   Q IS NOW AN ORTHOGONAL TRANSFORMATION MATRIX
      DO 40  I=1,N
      DO 40  J=1,N
   40 QT(I,J)=Q(J,I)
C   QT IS NOW THE TRANSPOSE,AND THE INVERSE, OF Q
      CALL MULT(SUB1,Q,N,SUB2,N2)
      CALL MULT(QT,SUB2,N,SUB1,N2)
C   SUB1 NOW CONTAINS THE TRIANGULAR MATRIX QT*A*Q
C      WHERE 'A' REPRESENTS THE INPUT VALUE OF SUB1.
      DO 50  I=1,N
      DO 50  J=1,N
   50 SUB2(I,J)=0.
C   SUB2 IS SET TO A ZERO MATRIX FOR INPUT TO MODULE FUNCT.
      CALL FUNCT(1,N,SUB1,SUB2,N2)
C   SUB2 NOW HOLDS EXP(A*TIME) IN TRIANGULAR FORM
      CALL MULT(SUB2,QT,N,SUB1,N2)
      CALL MULT(Q,SUB1,N,SUB2,N2)
C   SUB2 NOW HOLDS EXP(A*TIME) IN ORIGINAL BASIS FORM
      RETURN
      END
..
```

```
*DECK INTEG2
      SUBROUTINE INTEG(N,A,T,QUE,PHI,QD,QDT,N2)
C  FUNCTION- USES SIMPSON'S RULE ON THE RICCATI INTEGRAL.
C        - USES THE MATRIX RULE EXP(A(S+T))=EXP(AS)*EXP(AT).
C   SOURCE: A.H.LEVIS, COMPUTATIONAL ASPECTS OF THE MATRIX
C      EXPONENTIAL, IEEE TRANSACTIONS ON AUTOMATIC CONTROL,
C      AUG 1969:410,411.
C   INPUT VALUES:
C      A: SYSTEM MATRIX TO BE EXPONENTIATED.
C      T: TIME INCREMENT IN ARBITRARY UNITS.
C      QUE: SYSTEM MODEL ERROR MATRIX.
C      PHI,QD,QDT: STORAGE ARRAYS. WILL BE DESTROYED.
C      N2: GLOBAL LEADING DIMENSION OF ALL ARRAYS.
C      N: DIMENSION OF SUBMATRICES TO BE MANIPULATED.
C   OUTPUT VALUES:
C      N,T,N2: INPUT VALUES ARE UNCHANGED.
C      A,PHI,QD: INPUT CONTENTS ARE DESTROYED.
C      QDT: HOLDS THE MAIN RESULT,
C         INTEGRAL(EXP(-A*T)*QUE*EXP(-A'*T)DT)
      DIMENSION A(N2,N2),QUE(N2,N2),PHI(N2,N2)
      DIMENSION QD(N2,N2),QDT(N2,N2)
      T2= -T*0.5
      CALL MEXP(N,A(1,1),T2,PHI(1,1),QD(1,1),QDT(1,1),N2)
C   NOW PHI = EXP(-A*T/2)
      CALL TRI(N,QUE(1,1),PHI(1,1),QD(1,1),QDT(1,1),N2)
C   NOW QDT = EXP(-A*T/2)*QUE*EXP(-A'*T/2)
      CALL MSCALE(N,N,QDT(1,1),4.0,QDT(1,1))
C   NOW QDT = 4*EXP(-A*T/2)*QUE*EXP(-A'*T/2)
      CALL MULT(PHI(1,1),PHI(1,1),N,A(1,1),N2)
C   NOW A = EXP(-A*T)
      CALL TRI(N,QUE(1,1),A(1,1),QD(1,1),PHI(1,1),N2)
C   NOW PHI = EXP(-A*T)*QUE*EXP(-A'*T)
      DO 10 I=1,N
      DO 10 J=1,N
   10 QDT(I,J)= QUE(I,J)+ QDT(I,J)+ PHI(I,J)
C   QDT SUMMED THE THREE TERMS OF SIMPSON'S RULE.
      T6=T/6.0
      CALL MSCALE(N,N,QDT(1,1),T6,QDT(1,1))
C   NOW QDT = QDT*T/6. SIMPSON'S RULE IS COMPLETE.
      RETURN
      END
..
```

```
*DECK MULT2
      SUBROUTINE MULT2(N,X,Y,Z,N2)
C   FUNCTION- PERFORMS MATRIX MULTIPLICATION Z=X*Y',
C      WHERE Y' REPRESENTS THE TRANSPOSE OF MATRIX Y.
C   INPUT VALUES:
C      X,Y: ARRAYS THAT HOLD MATRICES TO BE MULTIPLIED.
C      Z:   STORAGE ARRAY. CONTENTS WILL BE DESTROYED.
C      N2:  THE GLOBAL LEADING DIMENSION OF ARRAYS X,Y,Z
C      N:   THE ORDER OF THE SUBMATRICES TO BE MULTIPLIED.
      DIMENSION X(N2,N2),Y(N2,N2),Z(N2,N2)
      DO 20  I=1,N
      DO 20  J=1,N
      Z(I,J)= 0.
      DO 20  K=1,N
   20 Z(I,J)= Z(I,J)+X(I,K)*Y(J,K)
      RETURN
      END
..
```

```
*DECK TRI
      SUBROUTINE TRI(N,Q,X,W,Z,N2)
C   FUNCTION- PERFORMS THE MATRIX MULTIPLICATION Z=X*Q*X',
C      WHERE X' REPRESENTS THE TRANSPOSE OF MATRIX X.
C   INPUT VALUES:
C      X,Q: ARRAYS THAT HOLD MATRICES TO BE MULTIPLIED.
C      W,Z: STORAGE ARRAYS. CONTENTS WILL BE DESTROYED.
C      N2:  THE GLOBAL LEADING DIMENSION OF ARRAYS Q,X,W,Z.
C      N:   THE DIMENSION OF SUBMATRICES TO BE MULTIPLIED.
C   OUTPUT VALUES:
C      X,Q,N2,N: INPUT VALUES ARE UNCHANGED.
C      Z:        HOLDS THE MAIN RESULT X*Q*X'.
C      W:        INPUT VALUES ARE DESTROYED.
      DIMENSION Q(N2,N2),X(N2,N2),W(N2,N2),Z(N2,N2)
      CALL MULT(X,Q,N,W,N2)
C   X*Q IS STORED IN W
      CALL MULT2(N,W,X,Z,N2)
C   Z=W*X"
      RETURN
      END
..
```

```
*DECK MULT
      SUBROUTINE MULT(X,Y,N,Z,N2)
C    FUNCTION- PERFORMS MATRIX MULTIPLICATION Z=X*Y
C    INPUT VALUES:
C       X,Y: ARRAYS THAT HOLD MATRICES TO BE MULTIPLIED
C       Z:   STORAGE ARRAY. CONTENTS WILL BE DESTROYED.
C       N2:  GLOBAL LEADING DIMENSION OF ARRAYS X,Y,Z.
C       N:   THE ORDER OF THE SUBMATRICES TO BE MULTIPLIED.
C    OUTPUT VALUES:
C       Z:   HOLDS THE MATRIX PRODUCT X*Y.
C       X,Y,N,N2: INPUT VALUES ARE UNCHANGED.
      DIMENSION X(N2,N2),Y(N2,N2),Z(N2,N2)
      DO 20  I=1,N
      DO 20  J=1,N
      Z(I,J)=0.
      DO 20  K=1,N
   20 Z(I,J)=Z(I,J)+X(I,K)*Y(K,J)
      RETURN
      END
..
```

```
*DECK FUNCT
      SUBROUTINE FUNCT(R,S,T,F,MM)
C   FUNCTION-THIS MODULE COMPUTES THE EXPONENT OF AN
C      UPPER TRIANGULAR MATRIX WITH DISTINCT EIGENVALUES.
C      THE EXPONENTIAL OF THE DIAGONAL ELEMENTS IS COMPUTED
C      DIRECTLY.SUPERDIAGONAL ELEMENTS ARE OBTAINED FROM THE
C      LOWER DIAGONALS BY A RECURRENCE RELATION.
C      A WARNING IS PRINTED IF MULTIPLE EIGENVALUES EXIST.
C   SOURCE: COMPUTATION OF FUNCTIONS OF TRIANGULAR MATRICES
C      BY B.N. PARLETT, MEMO #ERL-M481, UNIVERSITY OF CALIF.
C      AT BERKELEY. NOV 1974. (AD-A 005 916)
C   INPUT VALUES:
C      R: THE INDEX OF THE FIRST ROW IN THE TRIANGULAR BLOCK.
C      S: THE INDEX OF THE LAST ROW IN THE TRIANGULAR BLOCK.
C      T: THE ARRAY WHICH CONTAINS THE TRIANGULAR MATRIX.
C      F: THIS ARRAY CONTAINS A ZERO MATRIX.
C      MM: THE GLOBAL LEADING DIMENSION OF ARRAYS T AND F.
C   OUTPUT VALUES:
C      R,S,T,MM: INPUT VALUES ARE UNCHANGED.
C      F: CONTAINS THE RESULT, THE EXPONENTIAL OF MATRIX T.
      DIMENSION T(MM,MM),F(MM,MM)
      INTEGER R,S
      REAL EXP
      DO 10  I=R,S
C THE IF-BLOCK GIVES 14-DIGIT ACCURACY WITHOUT UNDERFLOW
      IF( T(I,I) .LT. -43.) THEN
         F(I,I)=0.
      ELSE
        F(I,I)=EXP( T(I,I) )
      END IF
   10 CONTINUE
C   PROCESS THE KTH SUPERDIAGONAL
      N=S-R+1
      NN=N-1
C   NN = NUMBER OF SUPERDIAGONALS IN THE BLOCK
      IF(NN .EQ. 0) RETURN
      DO 13  K=1,NN
      LL=S-K
         DO 12  I=R,LL
C        CHECK FOR MULTIPLE EIGENVALUES.
         DIFF=T(I,I)-T(I+K,I+K)
         IF(ABS(DIFF) .EQ. 0.0) GO TO 14
         G=T(I,I+K)*(F(I,I)-F(I+K,I+K))
         KK=K-1
         IF(KK .EQ. 0) GO TO 12
            DO 11  M=1,KK
   11  G=G+(F(I,I+M)*T(I+M,I+K)-T(I,I+K-M)*F(I+K-M,I+K))
   12  F(I,I+K)=G/DIFF
   13 CONTINUE
      RETURN
   14 PRINT *,ERROR IN MODULE FUNCT.'
      PRINT *,'TRIANGULAR MATRIX HAS MULTIPLE EIGENVALUES.'
      RETURN
      END
*DECK HQR
      SUBROUTINE HQR(IGH,H,Z,N2)
C   FUNCTION- CALCULATES THE ORTHOGONAL SCHUR TRANSFORMATION
C      MATRIX FOR THE UPPER HESSENBERG MATRIX WHICH IS INPUT.
C      USES THE QR ITERATIVE METHOD.
C      PRINTS A WARNING IF THE EIGENVALUES DO NOT CONVERGE.
C   SOURCE: THE EISPACK GUIDE.
C   INPUT VALUES:
C      IGH: DIMENSION OF THE REQUIRED TRANSFORMATION MATRIX.
C      H:   THE UPPER HESSENBERG MATRIX TO BE TRANSFORMED.
C      Z:   MUST BE AN IDENTITY MATRIX.
C      N2:  THE GLOBAL LEADING DIMENSION OF ARRAYS H AND Z.
C   OUTPUT VALUES:
C      IGH,N2: INPUT VALUES ARE UNCHANGED.
C      H:      INPUT VALUES ARE DESTROYED.
C      Z:      CONTAINS THE ORTHOGONAL SCHUR TRANSFORM MATRIX.
      INTEGER I,J,K,L,M,N,EN,II,JJ,LL,MM,NA,NM,NN,N2,
     X        IGH,ITS,LOW,MP2,ENM2,IERR,MINO
      REAL H(N2,N2),Z(N2,N2)
      REAL P,Q,R,S,T,W,X,Y,RA,SA,VI,VR,ZZ,NORM
      REAL MACHEP,SQRT,ABS,SIGN,REAL,AIMAG
      LOGICAL NOTLAS
      COMPLEX Z3,CMPLX
C
C
C
C   MACHEP IS A PARAMETER THAT SPECIFIES PRECISION
      MACHEP=0.0000000000001
      NM=IGH
      N=IGH
      LOW= 1
      IERR= 0
      NORM= 0.
      K= 1
C   COMPUTE MATRIX NORM
      DO 50  I = 1,N
         DO 40  J = K,N
   40    NORM = NORM + ABS(H(I,J))
         K = I
   50 CONTINUE
      EN = IGH
      T = 0.0
C     *** SEARCH FOR NEXT EIGENVALUES***
   60 IF(EN .LT. LOW) GO TO 1001
```

```
      ITS = 0
      NA = EN - 1
      ENM2 = NA - 1
C     ***LOOK FOR SINGLE SMALL SUB-DIAGONAL ELEMENT
C        FOR L=EN STEP -1 UNTIL LOW DO ***
   70 DO 80  LL = LOW, EN
         L = EN + LOW - LL
         IF(L .EQ. LOW) GO TO 100
         S = ABS(H(L-1,L-1)) + ABS(H(L,L))
      IF(S .EQ. 0.0) S = NORM
      IF(ABS(H(L,L-1)) .LE. MACHEP * S) GO TO 100
   80 CONTINUE
C     *** FORM SHIFT ***
  100 X = H(EN,EN)
      IF(L .EQ. EN) GO TO 270
      Y = H(NA,NA)
      W = H(EN,NA) * H(NA,EN)
      IF(L .EQ. NA) GO TO 280
      IF(ITS .EQ. 30) GO TO 1000
      IF(ITS .NE. 10 .AND. ITS .NE. 20) GO TO 130
C     *** FORM EXCEPTIONAL SHIFT ***
      T = T + X
      DO 120  I = LOW,EN
  120 H(I,I) = H(I,I) - X
      S = ABS(H(EN,NA)) + ABS(H(NA,ENM2))
      X = 0.75 * S
      Y = X
      W = -0.4375 * S * S
  130 ITS = ITS + 1
C       LOOK FOR TWO CONSECUTIVE SMALL SUB-DIAGONAL ELEMENTS
C     *** FOR M=EN-2 STEP -1 UNTIL L DO ***
      DO 140  MM = L, ENM2
         M = ENM2 + L - MM
         ZZ = H(M,M)
         R = X - ZZ
         S = Y - ZZ
         P = (R * S - W) / H(M+1,M) + H(M,M+1)
         Q = H(M+1,M+1) - ZZ - R - S
         R = H(M+2,M+1)
         S = ABS(P) + ABS(Q) + ABS(R)
         P = P/S
         Q = Q/S
         R = R/S
         IF(M .EQ. L) GO TO 150
         IF(ABS(H(M,M-1))*(ABS(Q) + ABS(R)) .LE. MACHEP*ABS(P)
     X    * (ABS(H(M-1,M-1))+ABS(ZZ)+ABS(H(M+1,M+1))))
     X    GO TO 150
  140 CONTINUE
  150 MP2 = M + 2
        DO 160  I = MP2, EN
           H(I,I-2) = 0.0
           IF(I .EQ. MP2) GO TO 160
           H(I,I-3) = 0.0
  160 CONTINUE
C   * DOUBLE QR STEP FOR ROWS L TO EN AND COLUMNS M TO EN *
        DO 260  K = M, NA
           NOTLAS = K .NE. NA
           IF (K .EQ. M) GO TO 170
           P = H(K,K-1)
           Q = H(K+1,K-1)
           R = 0.0
           IF (NOTLAS) R = H(K+2,K-1)
           X = ABS(P) + ABS(Q) + ABS(R)
           IF(X .EQ. 0.0) GO TO 260
           P = P/X
           Q = Q/X
           R = R/X
  170 S = SIGN(SQRT(P*P+Q*Q+R*R),P)
           IF(K.EQ. M) GO TO 180
           H(K,K-1) = -S * X
           GO TO 190
  180      IF(L .NE. M) H(K,K-1) = -H(K,K-1)
  190      P = P + S
           X = P/S
           Y = Q/S
           ZZ = R/S
           Q = Q/P
           R = R/P
C      *** ROW MODIFICATION ***
           DO 210  J = K, N
              P = H(K,J) +Q * H(K+1,J)
              IF(.NOT. NOTLAS) GO TO 200
              P = P + R * H(K+2,J)
              H(K+2,J) = H(K+2,J) - P * ZZ
  200         H(K+1,J) = H(K+1,J) - P * Y
              H(K,J) = H(K,J) - P * X
  210      CONTINUE
           J = MIN0(EN,K+3)
C      *** COLUMN MODIFICATION ***
           DO 230  I = 1, J
              P = X * H(I,K) + Y * H(I,K+1)
```

```
                IF( .NOT. NOTLAS) GO TO 220
                P = P + ZZ * H(I,K+2)
                H(I,K+2) = H(I,K+2) - P * R
  220           H(I,K+1) = H(I,K+1) - P * Q
                H(I,K) = H(I,K) - P
  230        CONTINUE
C       *** ACCUMULATE TRANSFORMATIONS ***
             DO 250  I = LOW, IGH
                P = X * Z(I,K) + Y * Z(I,K+1)
                IF( .NOT. NOTLAS) GO TO 240
                P = P + ZZ * Z(I,K+2)
                Z(I,K+2) = Z(I,K+2) - P * R
  240           Z(I,K+1) = Z(I,K+1) - P * Q
                Z(I,K) = Z(I,K) - P
  250        CONTINUE
  260   CONTINUE
        GO TO 70
C       *** ONE ROOT FOUND ***
  270   H(EN,EN) = X + T
        EN = NA
        GO TO 60
C       *** TWO ROOTS FOUND ***
  280   P = (Y - X)/ 2.0
        Q = P*P + W
        ZZ = SQRT(ABS(Q))
        H(EN,EN) = X + T
        X = H(EN,EN)
        H(NA,NA) = Y + T
        IF(Q .LT. 0.0) GO TO 320
C       *** REAL PAIR ***
        ZZ = P + SIGN(ZZ,P)
        X = H(EN,NA)
        S = ABS(X) + ABS(ZZ)
        P = X / S
        Q = ZZ / S
        R = SQRT(P*P + Q*Q)
        P = P / R
        Q = Q / R
C       *** ROW MODIFICATION ***
        DO 290  J = NA, N
           ZZ = H(NA,J)
           H(NA,J) = Q*ZZ + P*H(EN,J)
           H(EN,J) = Q*H(EN,J) - P*ZZ
  290   CONTINUE
C       *** COLUMN MODIFICATION ***
```

```
C      *** COLUMN MODIFICATION ***
       DO 300  I = 1, EN
          ZZ = H(I,NA)
          H(I,NA) = Q*ZZ + P*H(I,EN)
          H(I,EN) = Q*H(I,EN) - P*ZZ
  300  CONTINUE
C      *** ACCUMULATE TRANSFORMATIONS ***
       DO 310  I = LOW, IGH
          ZZ = Z(I,NA)
          Z(I,NA) = Q*ZZ + P*Z(I,EN)
          Z(I,EN) = Q*Z(I,EN) - P*ZZ
  310  CONTINUE
       GO TO 330
C      *** COMPLEX PAIR ***
  320  CONTINUE
  330  EN = ENM2
       GO TO 60
C ERROR - NO CONVERGENCE TO EIGENVALUE AFTER 30 ITERATIONS
 1000  IERR = EN
       PRINT *,'EIGENVALUE',IERR,'DOES NOT CONVERGE.'
       PRINT *,'SUGGEST INCREASING MACHEP IN MODULE HQR'
 1001  RETURN
       END
..
```

```
*DECK SGEFA
      SUBROUTINE SGEFA(A,LDA,N,IPVT)
      INTEGER LDA,N,IPVT(1),INFO
      REAL A(LDA,1)
C
C   SGEFA FACTORS A REAL MATRIX BY GAUSSIAN ELIMINATION.
C   A WARNING IS PRINTED IF A ZERO EIGENVALUE IS FOUND.
C
C   ON ENTRY:
C   A: THE MATRIX TO BE FACTORED
C   LDA: THE GLOBAL LEADING DIMENSION OF THE ARRAY A
C   N: THE ORDER OF THE MATRIX TO BE FACTORED.
C  IPVT:STORAGE ARRAY. INPUT CONTENTS WILL BE DESTROYED.
C
C   ON RETURN
C   A: AN UPPER TRIANGULAR MATRIX AND THE MULTIPLIERS
C      WHICH WERE USED TO OBTAIN IT.
C   IPVT: AN INTEGER VECTOR OF PIVOT INDICES
C   THIS IS FROM LINPACK USER'S GUIDE,VERSION 08/14/78
      REAL T
      INTEGER ISAMAX,J,K,KP1,L,NM1
C
C   GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
      INFO= 0
      NM1= N-1
      IF(NM1 .LT. 1) GO TO 70
      DO 60  K=1,NM1
      KP1= K+1
C
C   FIND L = PIVOT INDEX
      L= ISAMAX(N-K+1,A(K,K),1)+K-1
      IPVT(K)= L
C
C   ZERO PIVOT IMPLIES THIS COLUMN IS TRIANGULARIZED
      IF(A(L,K) .EQ. 0.0E0) GO TO 40
C
C   INTERCHANGE IF NECESSARY
      IF(L .EQ. K) GO TO 10
      T=A(L,K)
      A(L,K)= A(K,K)
      A(K,K)= T
   10 CONTINUE
C
C   COMPUTE MULTIPLIERS
      T= -1.0E0/A(K,K)
      CALL SSCAL(N-K,T,A(K+1,K),1)
```

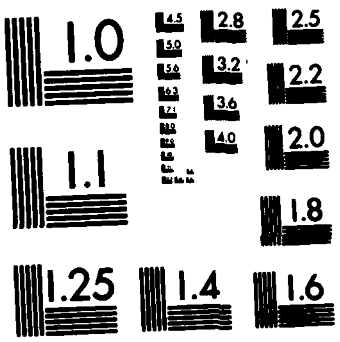
```
C
C   ROW ELIMINATION WITH COLUMN INDEXING
      DO 30  J=KP1,N
      T= A(L,J)
      IF(L .EQ. K) GO TO 20
      A(L,J)= A(K,J)
      A(K,J)= T
   20 CONTINUE
      CALL SAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
   30 CONTINUE
      GO TO 50
   40 CONTINUE
      INFO= K
   50 CONTINUE
   60 CONTINUE
   70 CONTINUE
      IPVT(N)= N
      IF(A(N,N) .EQ. 0.0E0) INFO= N
      IF(INFO .NE. 0) THEN
        PRINT *,'EIGENVALUE ',INFO,'=0 IN MODULE SGEFA.'
        PRINT *,'THIS CAUSES MODULE SGEDI TO DIVIDE BY 0.
      END IF
      RETURN
      END
*DECK SGEDI
      SUBROUTINE SGEDI(A,LDA,N,IPVT,WORK)
      INTEGER LDA,N,IPVT(1)
      REAL A(LDA,1),WORK(1)
C
C   SGEDI COMPUTES INVERSE OF MATRIX A USING
C    FACTORS COMPUTED BY SGEFA.
C
C   ON ENTRY:
C   A: THE OUTPUT FROM SGEFA, REAL(LDA,N)
C   LDA: THE LEADING DIMENSION OF ARRAY A
C   N: THE ORDER OF MATRIX A
C   IPVT: THE PIVOT VECTOR FROM SGEFA,INTEGER(N)
C   WORK: WORK VECTOR,CONTENTS DESTROYED,REAL(N)
C
C   ON RETURN:
C   A: INVERSE OF THE ORIGINAL MATRIX
C
C   ERROR CONDITION: A DIVISION BY ZERO WILL OCCUR IF THE
C   INPUT FACTOR CONTAINS A ZERO ON THE DIAGONAL.
C   IT WILL NOT OCCUR IF SGEFA HAS SET INFO=0
```

```
C
C     THIS IS FROM LINPACK USER'S GUIDE,VERSION 08/14/78
        REAL T
        INTEGER I,J,K,KB,KP1,L,NM1
C
C     COMPUTE INVERSE
        DO 100  K=1,N
        A(K,K)= 1.0E0/A(K,K)
        T= -A(K,K)
        CALL SSCAL(K-1,T,A(1,K),1)
        KP1= K+1
        IF(N .LT. KP1) GO TO 90
        DO 80  J=KP1,N
        T= A(K,J)
        A(K,J)= 0.0E0
        CALL SAXPY(K,T,A(1,K),1,A(1,J),1)
   80   CONTINUE
   90   CONTINUE
  100   CONTINUE
C
C     FORM INVERSE(U)*INVERSE(L)
        NM1= N-1
        IF(NM1 .LT. 1) GO TO 140
        DO 130  KB=1,NM1
        K= N-KB
        KP1= K+1
        DO 110  I=KP1,N
        WORK(I)= A(I,K)
        A(I,K)= 0.0E0
  110   CONTINUE
        DO 120  J=KP1,N
        T= WORK(J)
        CALL SAXPY(N,T,A(1,J),1,A(1,K),1)
  120   CONTINUE
        L= IPVT(K)
        IF(L .NE. K) CALL SSWAP(N,A(1,K),1,A(1,L),1)
  130   CONTINUE
  140   CONTINUE
        RETURN
        END
*DECK ISAMAX
        INTEGER FUNCTION ISAMAX(N,SX,INCX)
C     ISAMAX FINDS INDEX OF ELEMENT WITH MAX. ABSOLUTE VALUE.
C     LINPACK USER'S GUIDE, VERSION 03/11/78
        REAL SX(1),SMAX
```

AD-A124 873 NUMERICAL COMPUTATION OF THE MATRIX RICCATI EQUATION FOR HEAT PROPAGATION..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. 2/2

UNCLASSIFIED P W SAGSTETTER DEC 82 AFIT/GCS/MA/82D-7 .F/G 22/3 NL

END
FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
      INTEGER I,INCX,IX,N
      ISAMAX= 0
      IF(N .LT. 1) RETURN
      ISAMAX= 1
      IF(N .EQ. 1) RETURN
      IF(INCX .EQ. 1) GO TO 20
C
C   CODE FOR INCREMENT NOT EQUAL TO 1
      IX= 1
      SMAX= ABS(SX(1))
      IX= IX+INCX
      DO 10  I=2,N
      IF(ABS(SX(IX)).LE.SMAX) GO TO 5
      ISAMAX= I
      SMAX= ABS(SX(IX))
    5 IX= IX+INCX
   10 CONTINUE
      RETURN
C
C   CODE FOR INCREMENT EQUAL TO 1
   20 SMAX= ABS(SX(1))
      DO 30  I=2,N
      IF(ABS(SX(I)).LE.SMAX) GO TO 30
      ISAMAX= I
      SMAX= ABS(SX(I))
   30 CONTINUE
      RETURN
      END
*DECK SAXPY
      SUBROUTINE SAXPY(N,SA,SX,INCX,SY,INCY)
C   CONSTANT TIMES A VECTOR PLUS A VECTOR.
C   USES UNROLLED LOOP FOR INCREMENTS= 1.
C   FROM LINPACK USER'S GUIDE,VERSION 03/11/78
      REAL SX(1),SY(1),SA
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
      IF(N .LE. 0) RETURN
      IF(SA .EQ. 0.0) RETURN
      IF(INCX .EQ. 1 .AND. INCY .EQ. 1) GO TO 20
C
C   CODE FOR UNEQUAL INCREMENTS OR FOR
C    EQUAL INCREMENTS NOT EQUAL TO 1
      IX= 1
      IY= 1
      IF(INCX.LT.0)  IX= (-N+1)*INCX +1
      IF(INCY.LT.0)  IY= (-N+1)*INCY +1
      DO 10  I=1,N
      SY(IY)= SY(IY)+ SA*SX(IX)
      IX= IX+INCX
      IY= IY+INCY
   10 CONTINUE
      RETURN
C
C    CODE FOR BOTH INCREMENTS EQUAL TO 1
C    CLEAN-UP LOOP
   20 M=MOD(N,4)
      IF(M .EQ. 0) GO TO 40
      DO 30  I=1,M
      SY(I)= SY(I)+ SA*SX(I)
   30 CONTINUE
      IF(N .LT. 4) RETURN
   40 MP1= M+1
      DO 50  I=MP1,N,4
      SY(I)= SY(I)+ SA*SX(I)
      SY(I+1)= SY(I+1)+ SA*SX(I+1)
      SY(I+2)= SY(I+2)+ SA*SX(I+2)
      SY(I+3)= SY(I+3)+ SA*SX(I+3)
   50 CONTINUE
      RETURN
      END
*DECK SSCAL
      SUBROUTINE SSCAL(N,SA,SX,INCX)
C    SCALES A VECTOR BY A CONSTANT.
C    USES UNROLLED LOOPS FOR INCREMENT EQUAL TO 1.
C    LINPACK USER'S GUIDE,VERSION 03/11/78
C
      REAL SA,SX(1)
      INTEGER I,INCX,M,MP1,N,NINCX
      IF(N .LE. 0) RETURN
      IF(INCX .EQ. 1) GO TO 20
C
C    CODE FOR INCREMENT NOT EQUAL TO 1
      NINCX= N*INCX
      DO 10  I=1,NINCX,INCX
      SX(I)= SA*SX(I)
   10 CONTINUE
      RETURN
C
C    CODE FOR INCREMENT EQUAL TO 1.
C    CLEAN-UP LOOP
   20 M= MOD(N,5)
```

```
      IF(M .EQ. 0) GO TO 40
      DO 30  I=1,M
      SX(I)= SA*SX(I)
   30 CONTINUE
      IF(N .LT. 5) RETURN
   40 MP1= M+1
      DO 50  I=MP1,N,5
      SX(I)= SA*SX(I)
      SX(I+1)= SA*SX(I+1)
      SX(I+2)= SA*SX(I+2)
      SX(I+3)= SA*SX(I+3)
      SX(I+4)= SA*SX(I+4)
   50 CONTINUE
      RETURN
      END
*DECK SSWAP
      SUBROUTINE SSWAP(N,SX,INCX,SY,INCY)
C   INTERCHANGES TWO VECTORS.
C   USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO 1.
C   LINPACK USER'S GUIDE,VERSION 03/11/78
C
      REAL SX(1),SY(1),STEMP
      INTEGER I,INCX,INCY,IX,IY,M,MP1,N
      IF(N .LE. 0) RETURN
      IF(INCX .EQ. 1 .AND. INCY .EQ. 1) GO TO 20
C  CODE FOR UNEQUAL INCREMENTS, OR EQUAL
C   INCREMENTS NOT EQUAL TO 1.
      IX= 1
      IY= 1
      IF(INCX .LT. 0)  IX= (-N+1)*INCX+1
      IF(INCY .LT. 0)  IY= (-N+1)*INCY+1
      DO 10  I=1,N
      STEMP= SX(IX)
      SX(IX)= SY(IY)
      SY(IY)= STEMP
      IX= IX+INCX
      IY= IY+INCY
   10 CONTINUE
      RETURN
C
```

```
C    CODE FOR BOTH INCREMENTS EQUAL TO 1.
C    CLEAN-UP LOOP
   20 M= MOD(N,3)
      IF(M .EQ. 0) GO TO 40
      DO 30  I=1,M
      STEMP= SX(I)
      SX(I)= SY(I)
      SY(I)= STEMP
   30 CONTINUE
      IF(N .LT. 3) RETURN
   40 MP1= M+1
      DO 50  I=MP1,N,3
      STEMP= SX(I)
      SX(I)= SY(I)
      SY(I)= STEMP
      STEMP= SX(I+1)
      SX(I+1)= SY(I+1)
      SY(I+1)= STEMP
      STEMP= SX(I+2)
      SX(I+2)= SY(I+2)
      SY(I+2)= STEMP
   50 CONTINUE
      RETURN
      END
```

VITA

Philip Wayne Sagstetter was born on 31 December 1949 in St. Paul, Minnesota. He graduated from Hill High School in 1967 and attended St. Mary's College in Winona from which, in August 1971, he received the degree of Bachelor of Arts with a major in Physics. In May 1972, he received a commission in the USAF through the OTS Program. He completed Navigator training and received his wings in March 1973. After receiving further training in the F-4 aircraft, he served for eight years as an F-4 Weapons System Officer and as a Wing staff expert in contingency plans for tactical fighter operations. During this time, he was assigned to bases in Thailand, Britain, Georgia and Korea. He served as a Flight Commander in the 497th Tactical Fighter Squadron at Taegu AB, Korea before entering the School of Engineering, Air Force Institute of Technology in June 1981.

Permanent Address: 1696 E. Third Street
St. Paul, Minnesota 55106

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/MA/82D-7	2. GOVT ACCESSION NO. AD-A124 873	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) NUMERICAL COMPUTATION OF THE MATRIX RICCATI EQUATION FOR HEAT PROPAGATION DURING SPACE SHUTTLE REENTRY		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Philip W. Sagstetter Capt		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December 1982
		13. NUMBER OF PAGES 100
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; IAW AFR 190-17

FREDERICK C. LYNCH, Major, USAF
Director of Information

18. SUPPLEMENTARY NOTES

Approved for public release: IAW AFR 19
[illegible]
LYNN E. WOLAVER
Dean for Research and Professional Develo
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

4 JAN 19

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Matrix Riccati Equation
Exponential Matrix
Schur Matrix Decomposition
Space Shuttle

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A computer program named HEATEST required excessive computer time to evaluate the matrix Riccati equation for temperature covariance. Alternative numerical methods were employed to compute the Riccati equation, and the HEATEST program execution time was reduced by 70%. However, cumulative temperature covariance rose from 2.45 to 3.28 degrees. This rise was considered insignificant.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

A survey was conducted of methods for computing the matrix exponential. A triangular matrix decomposition method proved to be more efficient than summing the Taylor series, especially for matrices with a large condition number. This substitution produced an overall 10% decrease in HEATEST execution time with comparable accuracy.

Simpson's Rule was used to evaluate the matrix Riccati integral term. The accuracy of this method was in the range of 5 to 9 significant digits, and computation time for the integral term was reduced by 90% for a matrix of order 13. This substitution prompted the rise in the covariance.

FORTRAN program modules and numerical examples are included.